ダウンロードして使える
サンプルゲーム11本

ゲームで学ぶ「JavaScript入門」

HTML5&CSSも身につく!

田中賢一郎[著]
Kenichiro Tanaka

初心者のためのラクラク参考書!

SCORE
0031640
excellent!!
TIME

↓↓↓
ブラウザゲームを作りながら学習する!

↓↓↓
PC/iPhone/Android で試せる!

対応ブラウザ Internet Explorer、Chrome、Safari、Edge

インプレス

ゲームで学ぶ「JavaScript入門」

HTML5&CSSも身につく!

田中賢一郎［著］
Kenichiro Tanaka

インプレス

はじめに

プログラミングの上達に近道はないと思っています。「いろんなコードを読んで引き出しを増やし、自分でいろいろ書いて試行錯誤し、技術や知識を定着させる」、このようなプロセスを繰り返すのが一番だと感じています。

筆者が子どものころは、街の書店に数多くのパソコン雑誌が並んでいました。当時小学生だった自分は"ゲームをやりたい"という一心で、雑誌に掲載されているソースをひたすら入力したものでした。もちろん、言語を体系立てて習ったことはなく、わけもわからず入力し、デバッグや改造を繰り返していただけですが、そこから得たものは非常に大きかったように思います。最近は高品位なゲームも無料でダウンロードできるようになり、自分でソースを入力する必要性はなくなってしまいました。

本書に掲載したプログラムはダウンロード可能です。しかしながら、時間に余裕がある方はぜひ自分の手で入力してみてください。おそらく、一度で動くことはないでしょう。デバッグを通して何かしら学びがあるはずです。動くようになったら改造してみましょう。表示を変えたり、機能を追加したり、いろいろ試してみることで一層理解が深められるはずです。

本書はHTML5/CSS/JavaScript/ゲームといった広い範囲をカバーしているため、難しく感じる箇所があるかもしれません。壁にぶつかったときは、まず自分で調べてみましょう。プログラミングは楽しいアクティビティです。身に着けておくと将来かならず役に立つときがきます。本書がきっかけとなり「ゲームをつくってみよう」と思っていただき、Webやブラウザ、プログラミングの基礎を身につけていただければ、この上ない喜びです。

2015年11月吉日　著者

ゲームで学ぶ「JavaScript入門」

◉目次◉

Chapter 5　実践:ゲームプログラミング――191

Chapter 6　物理エンジンを使ったゲーム――279

動作環境

本書はWindows 10のInternet Explorer 11、Edge (20.10240.16384.0)、Google Chrome (46.0.2490.86m) といったブラウザを使用してコンテンツ開発と動作検証を行いました。また、iPad (iOS 9.1) 上のSafari、Android (Zenfone2) 上のChromeでもひととおりサンプルが動作することを確認しています (2015年11月20日時点)。サンプルコンテンツはHTML/CSS/JavaScriptの基本的な内容で実装されているため、最近のブラウザであればプラットフォームを問わず動くと思います。しかしながら、OSやブラウザ、機種によっては挙動が異なることもあります。特にタッチに関しては未だ動作のばらつきが大きいため、意図した挙動にならない可能性があることご了承ください。

すべてのサンプルはWebサーバとの通信を必要としません。必要なファイルがローカルにあればインターネットに接続していない環境でも実行可能です。さらに、サードパーティのライブラリも一切使用していません。プログラミングの基本、楽しさといったところにフォーカスしたかったので、そのようなスタンスで開発・実装を進めました。

●サンプルプログラムについて

本書で解説したサンプルプログラムは、以下のURLからすべてダウンロードできます。

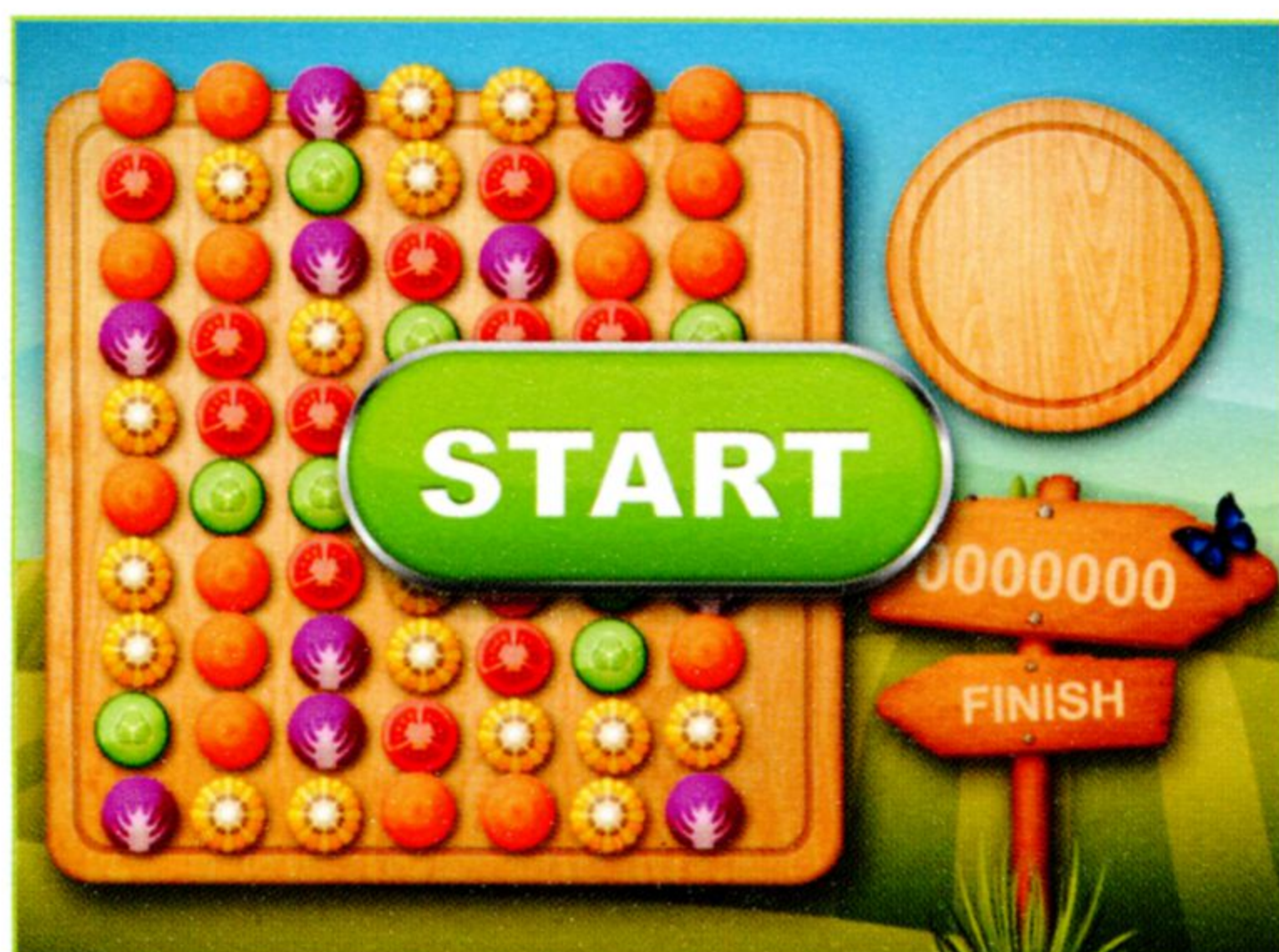
http://book.impress.co.jp/books/1115101084

本書でつくるサンプルゲーム

Chapter 1

HTML5
CSS
JavaScript
Canvas
Game
and
Physics engine

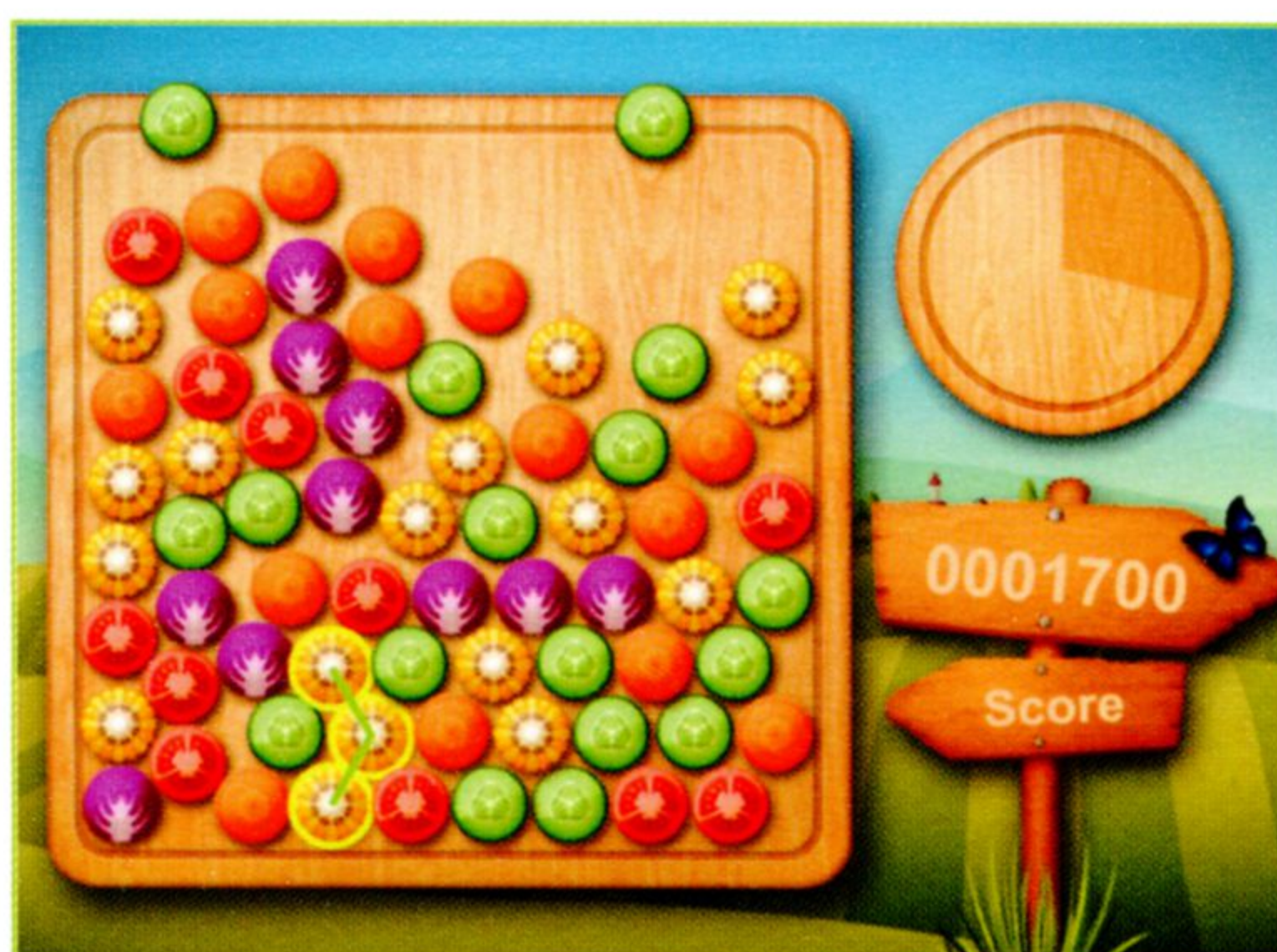
本書で解説されているゲームは、解説書にありがちな「説明のためのゲーム」ではなく、実際に遊んで楽しいもの、自作するときの参考になるような実用的なものを目指しました。各ゲームのソースコードは省略なく全文を掲載されています。つまり、行数はそれほど長くないのです。物事は楽しくないと続きません。本書も楽しみながら読み進んでいきましょう。

Vegetable March

遊び方

スタートボタンを押下すると軽快な音楽とともにゲームが始まります。近くにある同じ野菜を連結して消してください。消した分の野菜は上から落ちてきます。野菜がぶつかりながら落ちていく様子は見ているだけで楽しくなることでしょう。制限時間は約１分間、できるだけ多くの野菜を消して高得点を狙ってください。

開発のポイント

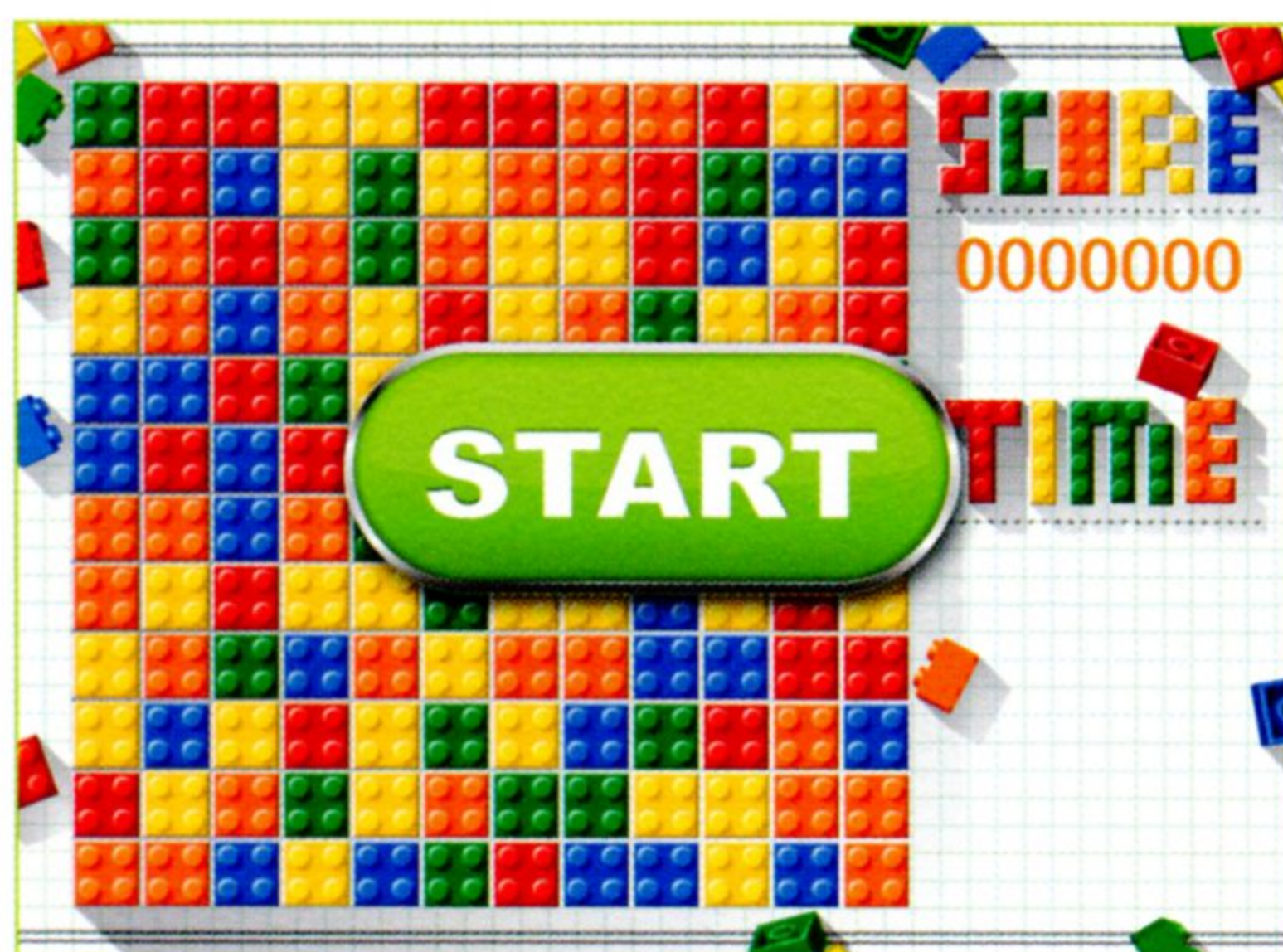
野菜がぶつかりながら落ちてくる様子は一見すると複雑そうに感じるかもしれませんが、ゲーム本体のソースコードは230行程度、本書用に開発した独自の2D物理エンジンTiny2D.jsのソースコードも260行程度にすぎません。この種のゲームの中ではかなり短いコードに抑えることができたと思っています。一般的なリアルタイムゲームと同じように、メインループでマウスやタッチのイベントを処理し、定期的にcanvasの画面を更新しています。

本ゲームのポイントはやはり物理エンジンです。一般的な物理エンジンライブラリは非常に高機能ですが、それだけに習得に時間がかかります。一方、本書で実装したエンジンは極限まで機能を削ったので260行程度しかありません。ほかのエンジンと比べると習得はもちろん、中身の理解や修正も容易なはずです。

このゲームで学ぶこと

- 物理世界のオブジェクトを画像を使って描画する
- 残り時間を扇形のゲージで描画する

学習のポイント

プログラミング初心者の方であれば、野菜や背景の画像をかえる、スコアの加算方法をかえる、BGMを変える、といったことを試してみてください。それだけでもゲームの印象は大きくかわるはずです。物理エンジンTiny2D.jsはわかりやすさを優先しています。パフォーマンスチューニングは行っていません。スキルに自信のある方はボトルネック部分を特定し、パフォーマンスの改善に取り組んでみてください。角速度のサポートなど新たな機能を追加しても面白いかもしれません。改善の余地はたくさんあるはずです。

⦿ ⦿ ⦿ Funky Blocks

遊び方

いわゆる落ち物系ゲームです。こちらも軽快なBGMと共にゲームがスタートします。同じ色が縦方向、もしくは横方向に3つ以上並ぶよう、タイルを上下・左右に入れ替えてください。ブロックが消えると上のブロックが落ちてきます。短時間に連続して消すと連鎖モードに突入しさらにスコアが加算されます。

開発のポイント

このゲームのソースコードは全部で330行程度です。デザイナーの力量によるところが大きいのですが、入門書の題材の割には高いクオリティを実現できたのではないかと思っています。個々のタイルはTileオブジェクトとして実装しています。タイルをスムーズに落下させる方法、タイルを消した時に表示されるメッセージをフェードアウトする方法、ゲーム開始時に同色が3つ並ばないようにする方法などに着目してください。

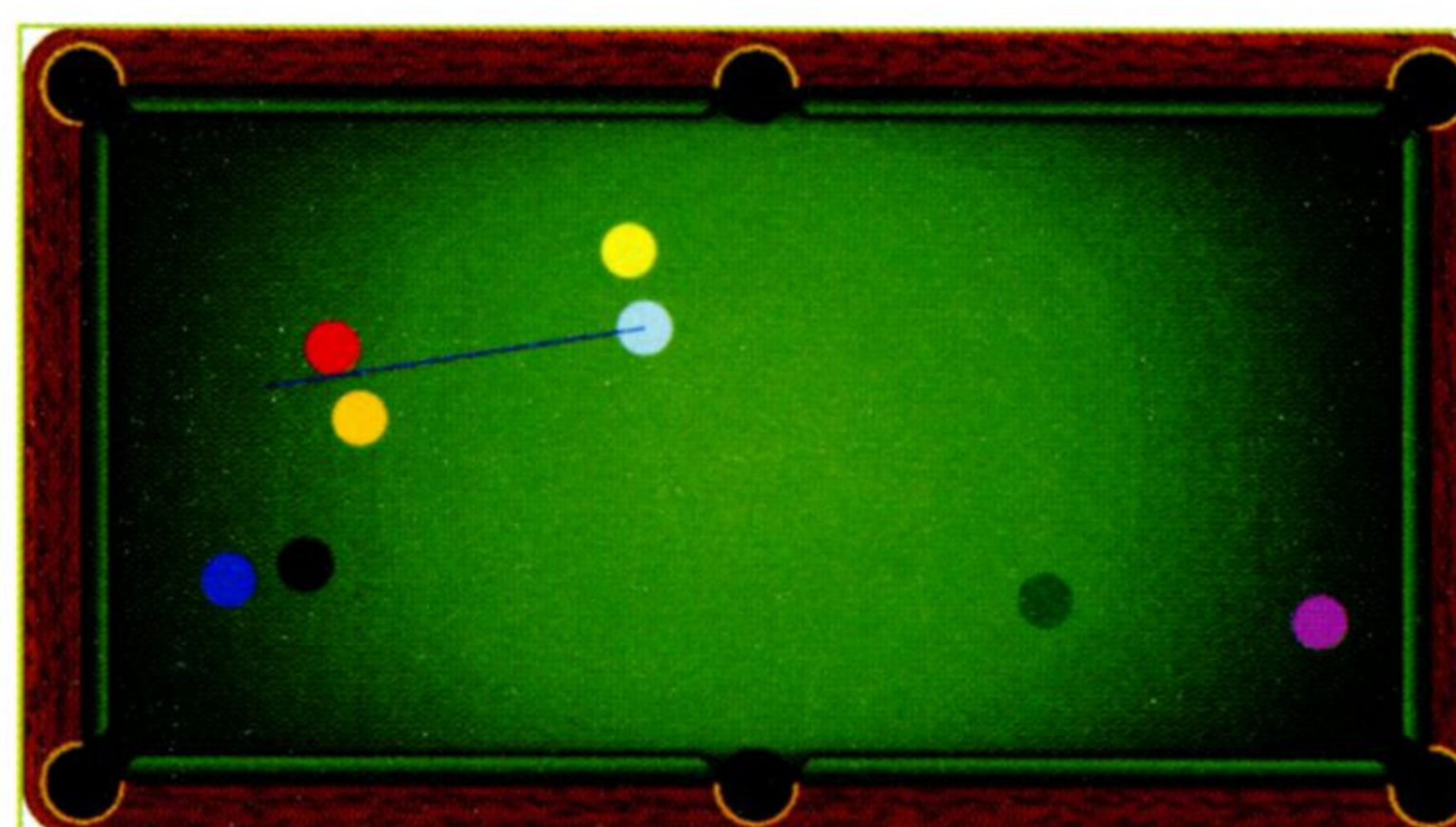
このゲームで学ぶこと

- ◉本格的なゲームを作る
- ◉効果音の再生方法を知る

学習のポイント

メインループを使った一般的なリアルタイムゲームです。特別な処理はありませんが、コードを短くするために関数オブジェクトを活用しています。そのあたりに着目してコードを読み進めていただければと思います。

背景画像を描画する、メッセージを工夫する、連鎖による加点を凝ったものにする、まだまだ改良の余地はたくさんあります。自分なりの工夫を加えることでより高い完成度のゲームに仕上げてください。

Billiard

遊び方

ご存じビリヤードです。ただし、実際にゲームのルールは実装していません。玉を突いて穴に落としてください。何回ですべての玉を落とせるか、もしくは実際のゲームと同じように2人で競ってもよいでしょう。

開発のポイント

このゲームもVegetable Marchと同じくTiny2D.jsを利用しています。このゲーム本体のソースコードは150行程度です。ポケットの穴は円オブジェクトとして実装されており、その円オブジェクトと衝突したときに玉を消去しています。物理世界にあるオブジェクトに初速度を与える方法、衝突時の処理方法など、物理エンジンの基本的な使い方を学習するには良い題材だと思います。

このゲームで学ぶこと

- マウスやタッチの座標が物理世界のオブジェクトに含まれるか検出する（手玉の検出）
- 物理世界のオブジェクトが衝突した時にオブジェクトを消去する（ポケットへ落下時）

その他

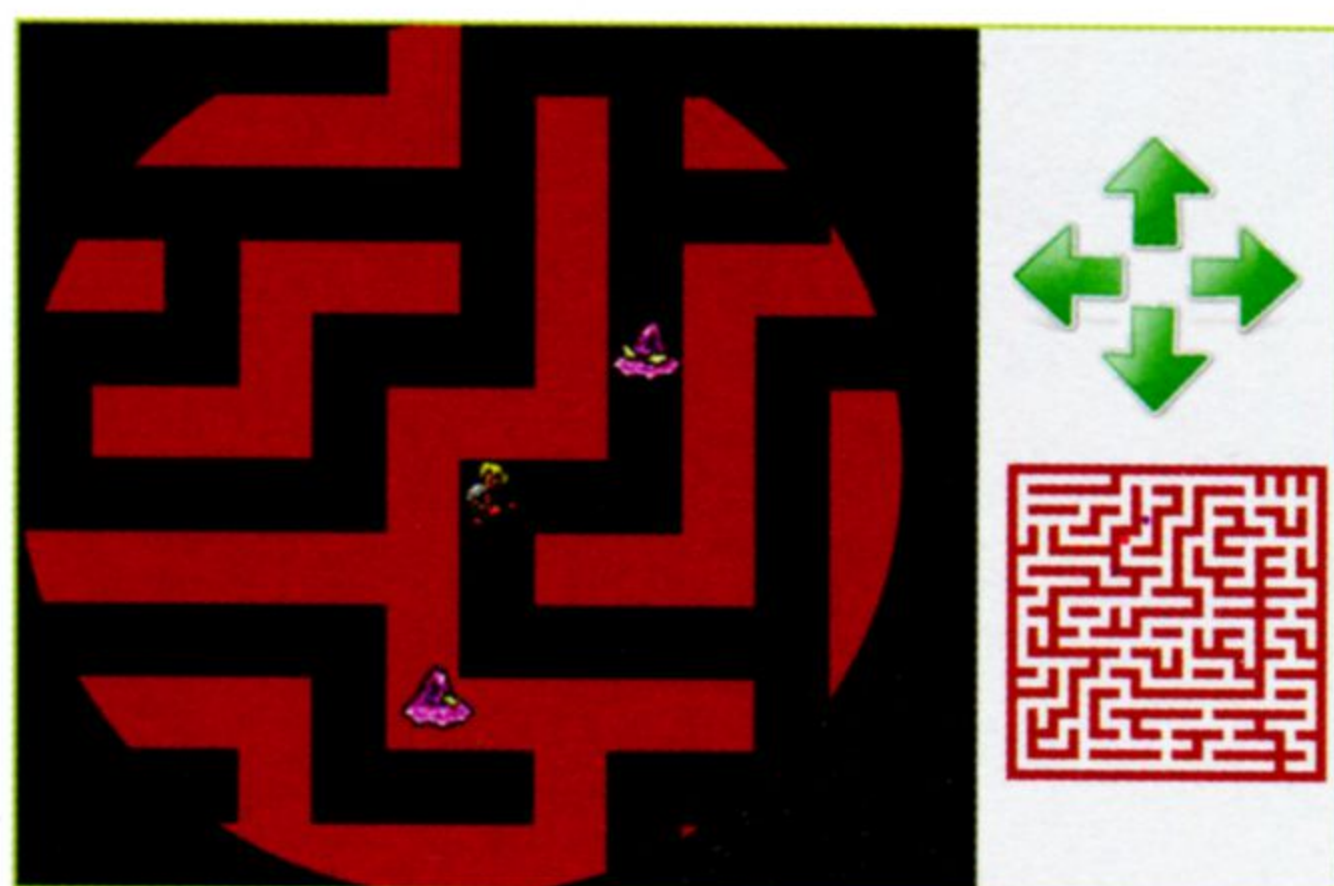
ビリヤードはあくまでも一例にすぎません。今回はビリヤードという題材にしましたが、背景をバトル場にして、玉の代わりにモンスターの画像を使ってみると…人気ゲームが思い浮かぶのではないでしょうか？ 画像を差し替えるだけでも印象はガラリと変わるはずです。さらに、障害物を配置する、爆発物を配置する、いろいろなアイディアを盛り込むことができるはずです。みなさんの想像力を働かせて面白いゲームに変貌させてください。オリジナルゲームをつくる楽しみを実感しましょう。

Saturn Voyager

隕石を避けてどこまで進めるかを競うゲームです。疑似3Dではありますが、宇宙空間を高速に進んでいるような視覚効果を表現できていると思います。このゲームのソースコードは170行程度です。隕石を自分からの距離に応じてソートし、遠い順から順番に描画することで自然な効果を演出する、ペインターのアルゴリズムを実装しています。また、canvasの座標系変換をつかって隕石を回転させています。画像の回転はいろいろなゲームで利用できるので覚えておきたい手法のひとつです。

このゲームで学ぶこと

- Canvasの座標軸変換になれる（画像の回転）
- 疑似3Dモデルに親しむ

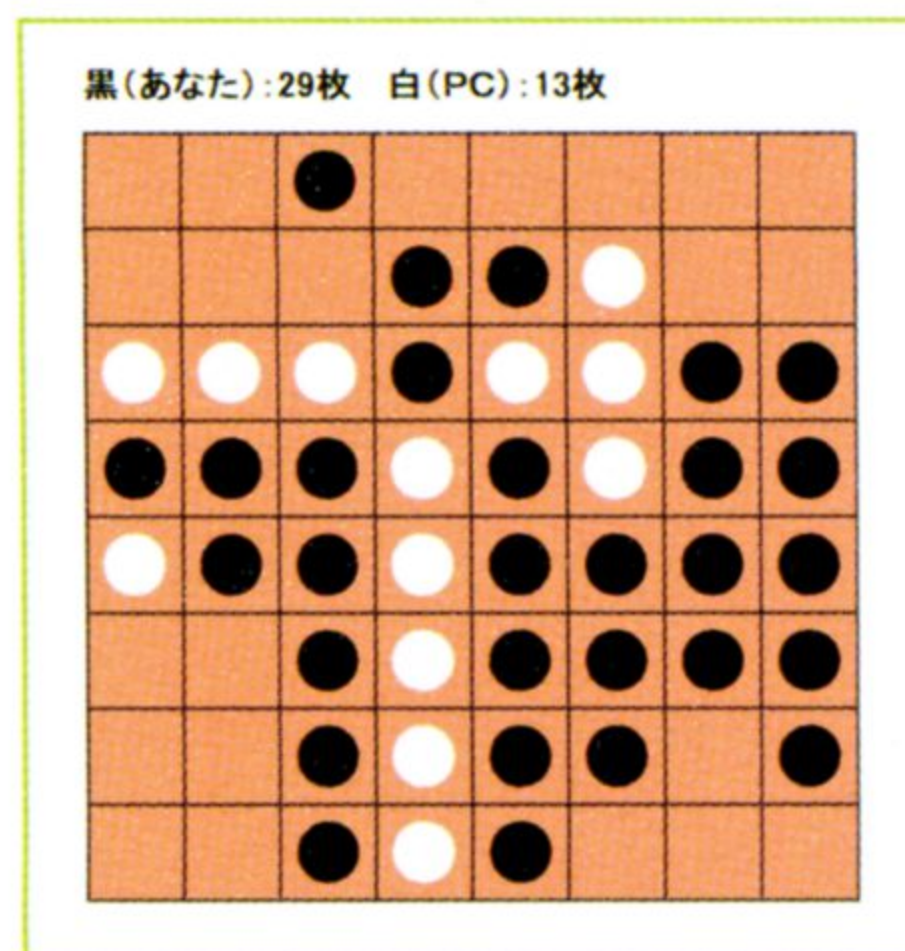
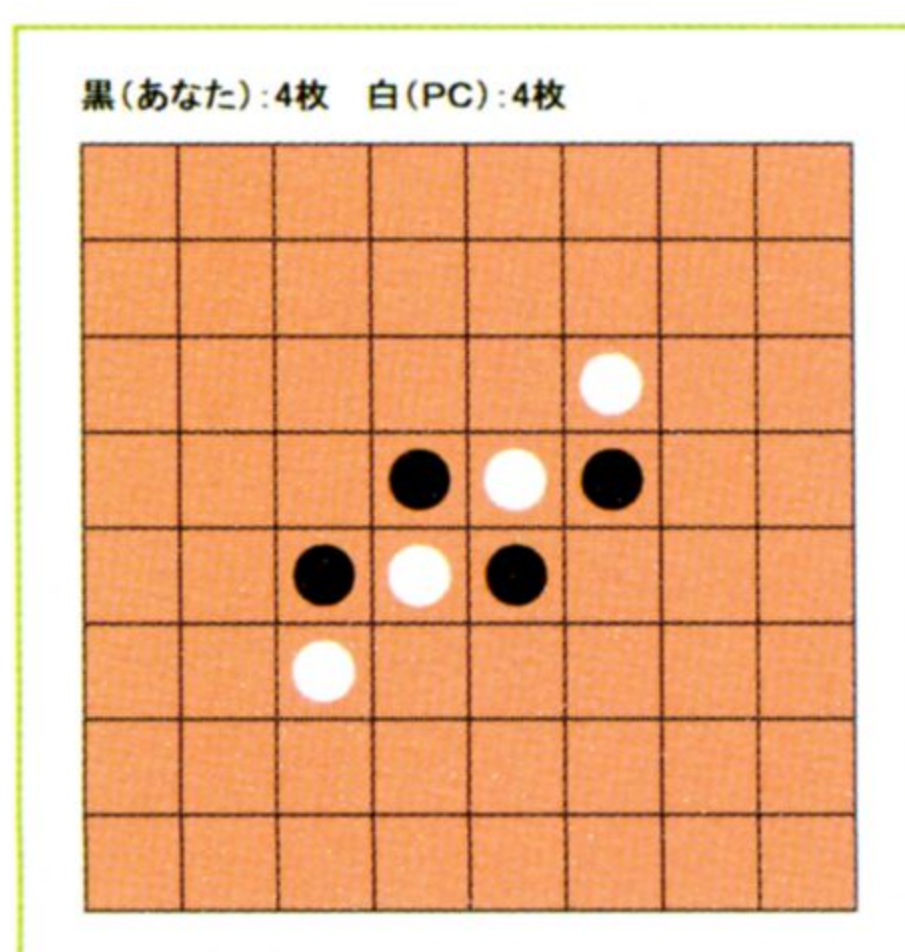
Dungeon

主人公を上下左右キーで操作して迷路右下のゴールを目指してください。モンスター２匹が容赦なく襲い掛かります。迷路は毎回動的に作成されます。画面右側には地図が表示されます。このゲーム本体のソースコードは340行程度です。迷路の生成には棒倒し法というアルゴリズムを使用しています。主人公の視野を表現するため、円形のクリップリージョンを設定して描画を行っています。移動もスムーズにスクロールするように工夫しています。

このゲームで学ぶこと

- 上下左右スクロールゲームに慣れる
- 迷路の自動生成を行う
- Canvasのリージョンクリップの手法を知る

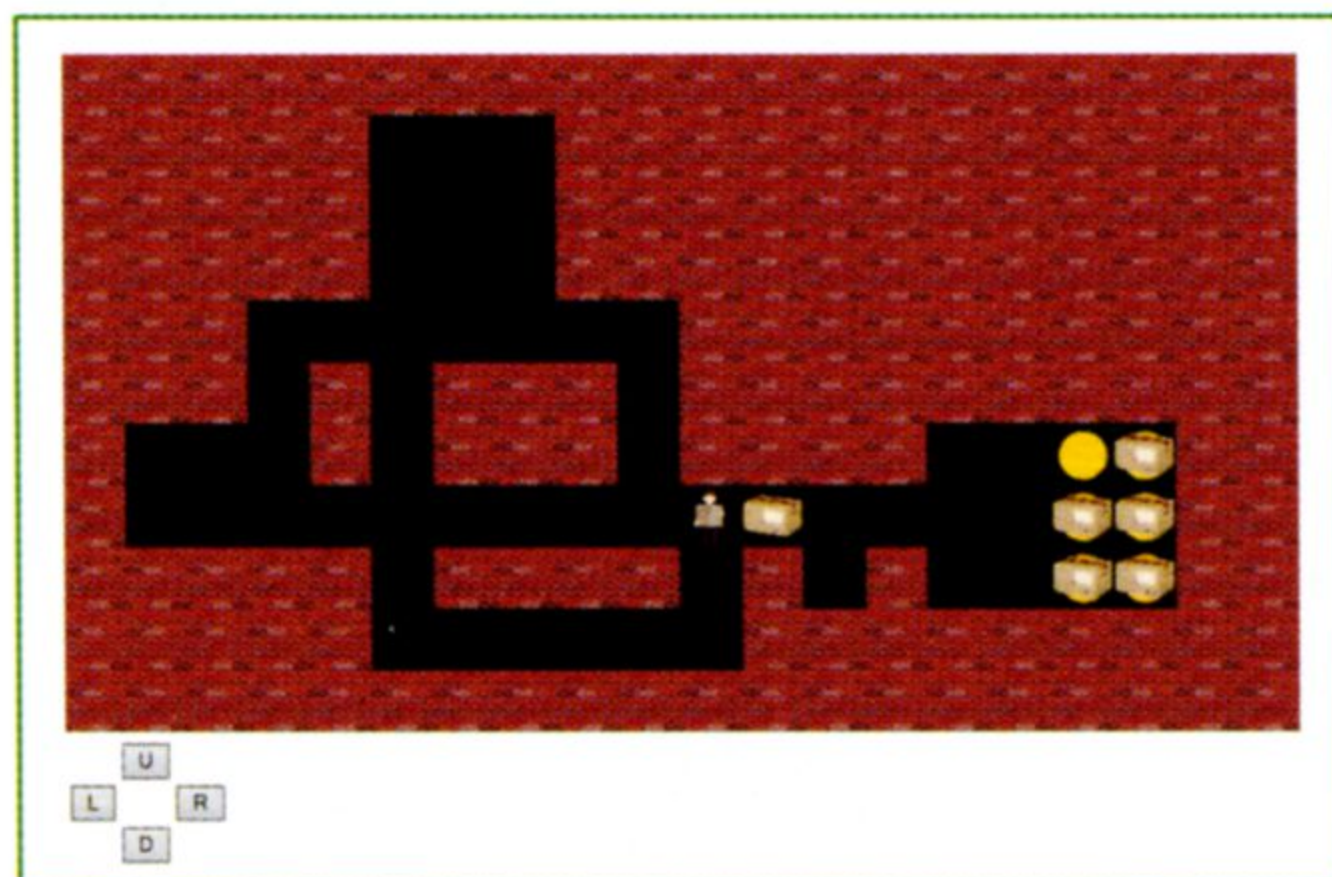
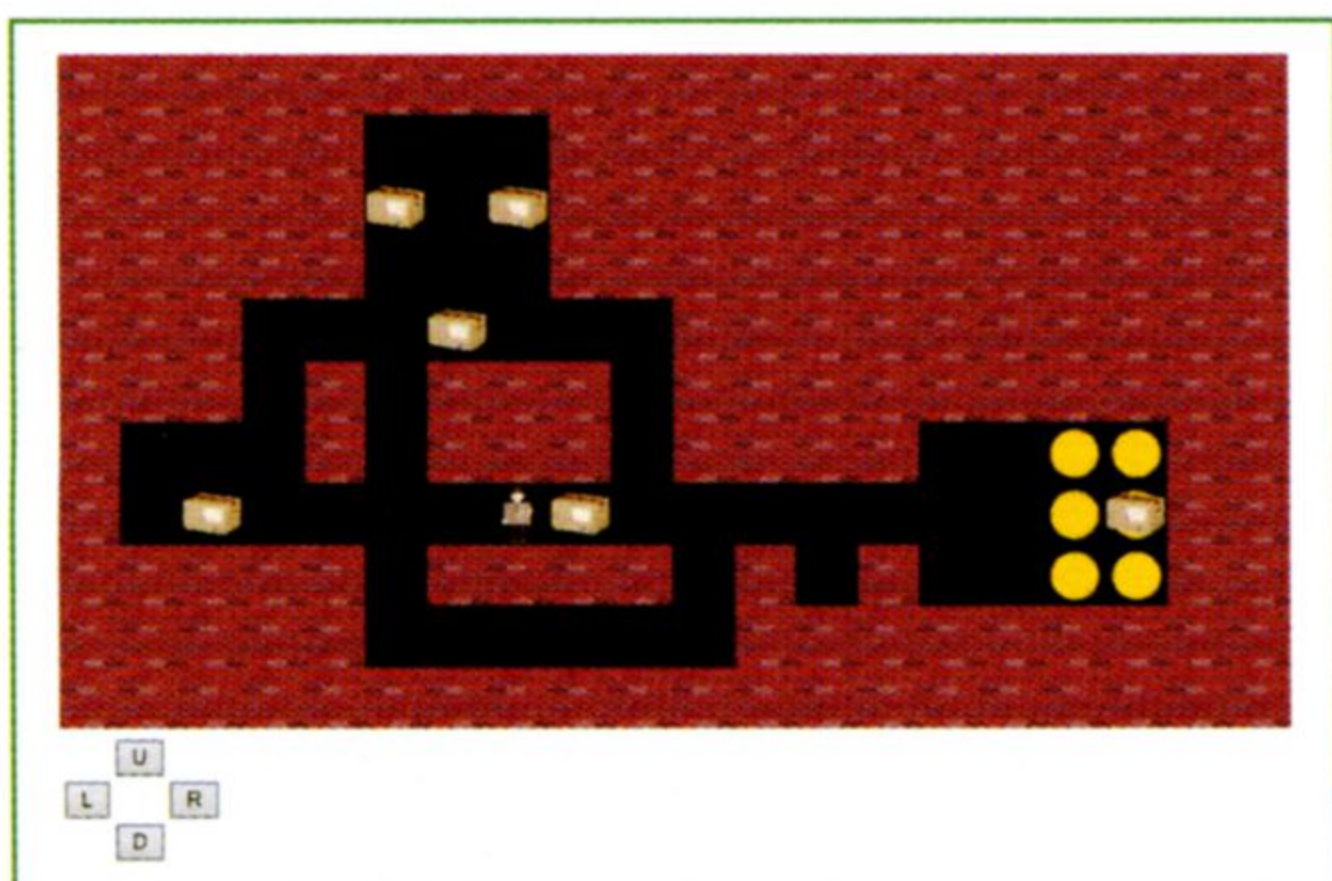
⦿ ⦿ ⦿ Reversible Piece

黒(あなた):4枚　白(PC):4枚

黒(あなた):29枚　白(PC):13枚

ルール説明は不要でしょう。石を置いて相手を挟んでひっくり返し、数の多いほうが勝ちというゲームです。ソースコードは280行程度です。コンピュータがあなたの対戦相手をしてくれます。このゲームは四隅を取ったほうが圧倒的に有利になることはご存知だと思います。それぞれのマス目に優先順位の重みづけを設定し、合計値が高くなるように石を置いていくというシンプルなアルゴリズムです。コンピュータの思考アルゴリズムがどのように実装されているか、その一例としてコードを読んでいただければと思います。

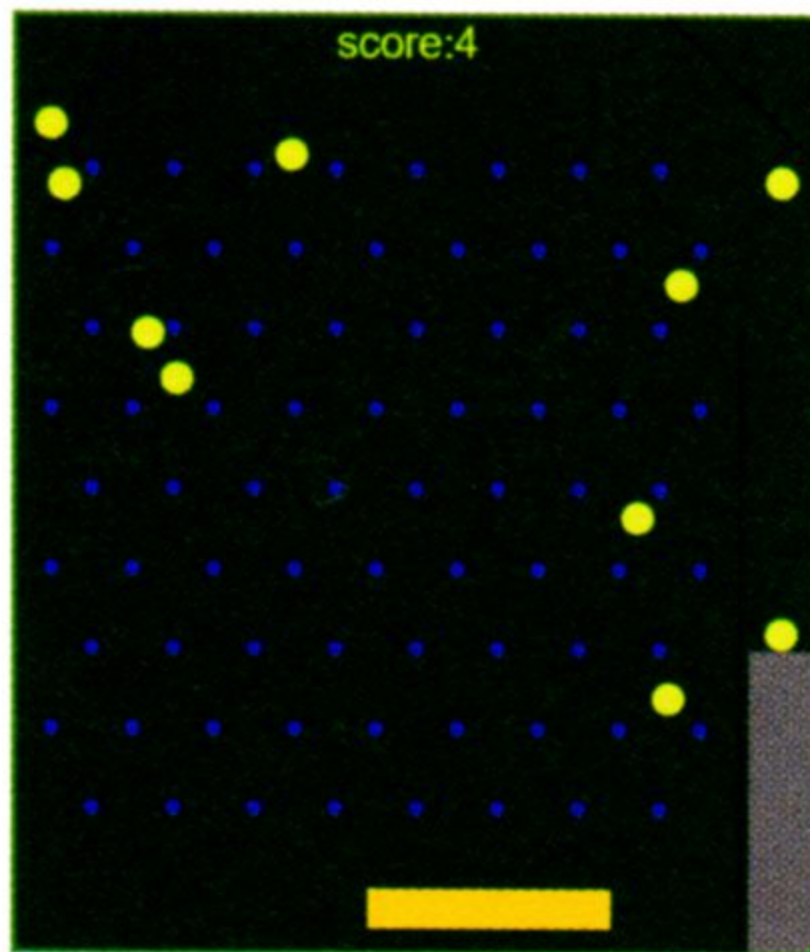
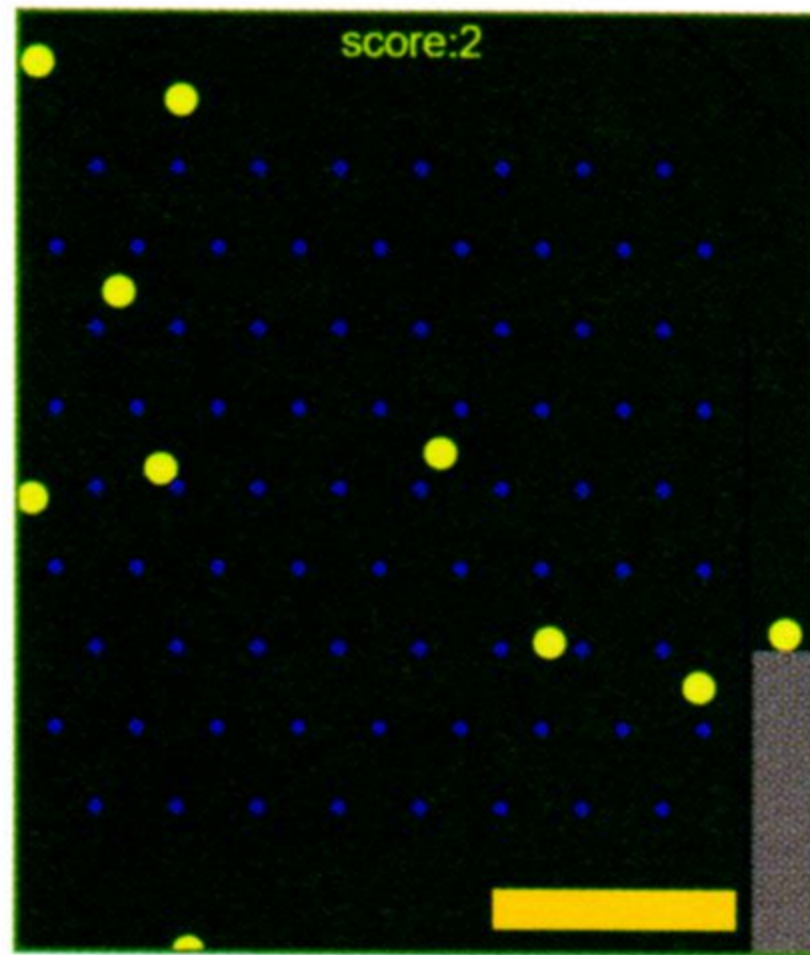
このゲームで学ぶこと

- ◉コンピュータの思考回路を実装する

⦿ ⦿ ⦿ Carry It

プレーヤーを操作してすべての荷物をゴール地点に運んでください。ただし、荷物を押すことはできても引っ張ることはできません。荷物を2つ同時に押すこともできません。実際にやってみるとパズル的な面白さを実感できると思います。地図データを含めてもコードは100行未満です。実際にコードをみるとその短さに驚かれるかもしれません。コードを短くするためビット演算という手法を使用しました。解説記事を参考に丁寧に読み進めてください。

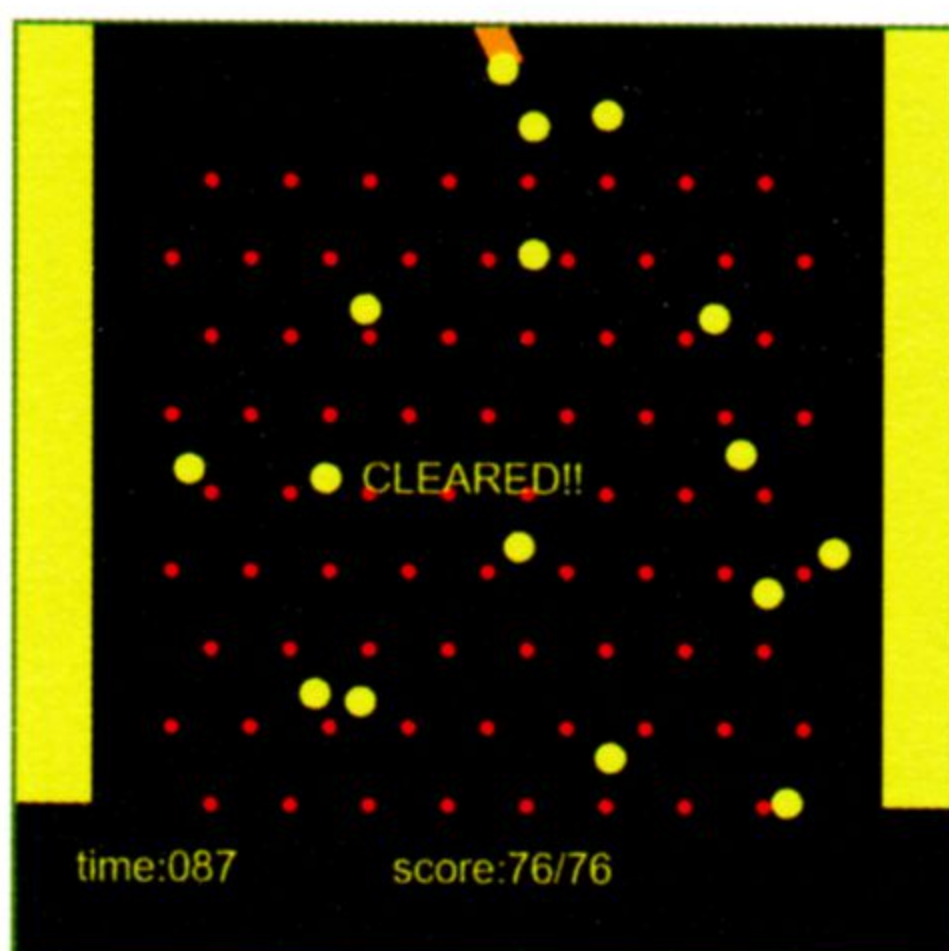
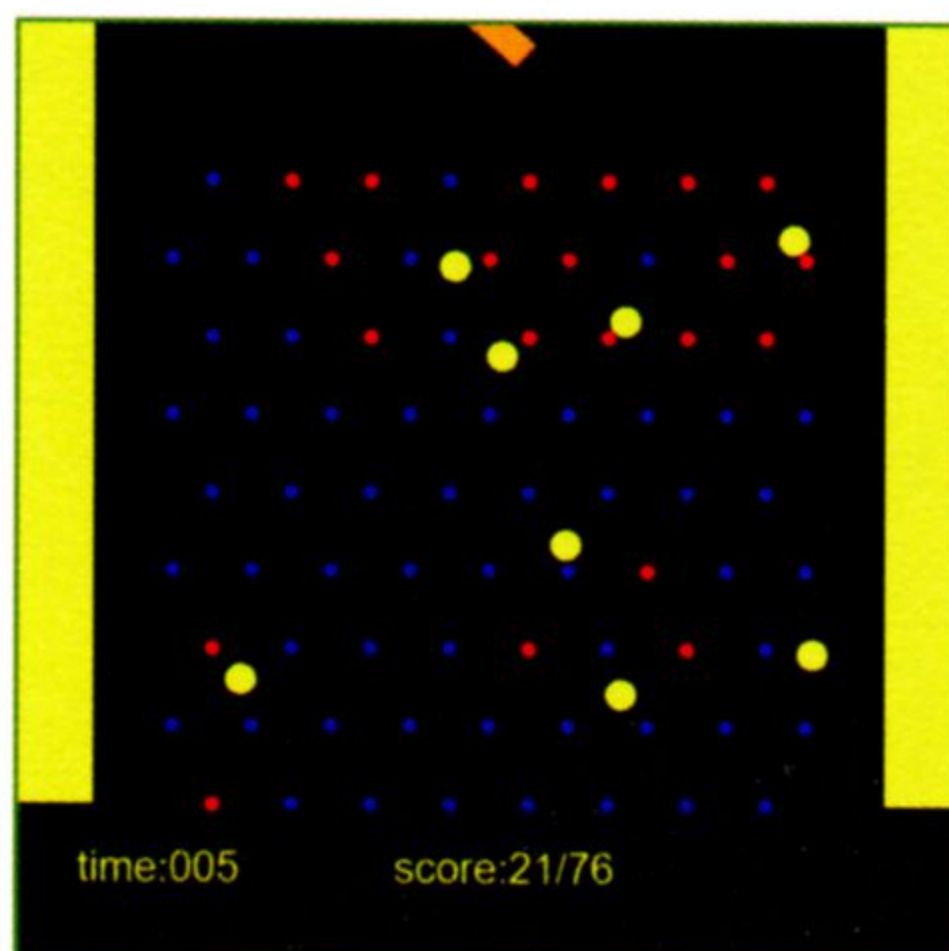
このゲームで学ぶこと

- ◉仮想マップの使い方になれる
- ◉JavaScriptから画像をcanvasに描画する
- ◉ビット演算になれる

Pachinko

画面上をクリック・タップしてレバーを引っ張り、パチンコ玉を発射してください。画面下部で左右に動いているターゲットに当たると得点です。ソースコードは150行程度です。物理エンジンでは、時計を進めるだけでオブジェクトの位置が自動的に計算されます。画面下部にあるターゲットだけはJavaScriptから明示的に場所を変更しています。JavaScriptから物理世界を直接制御する一例としてご覧ください。

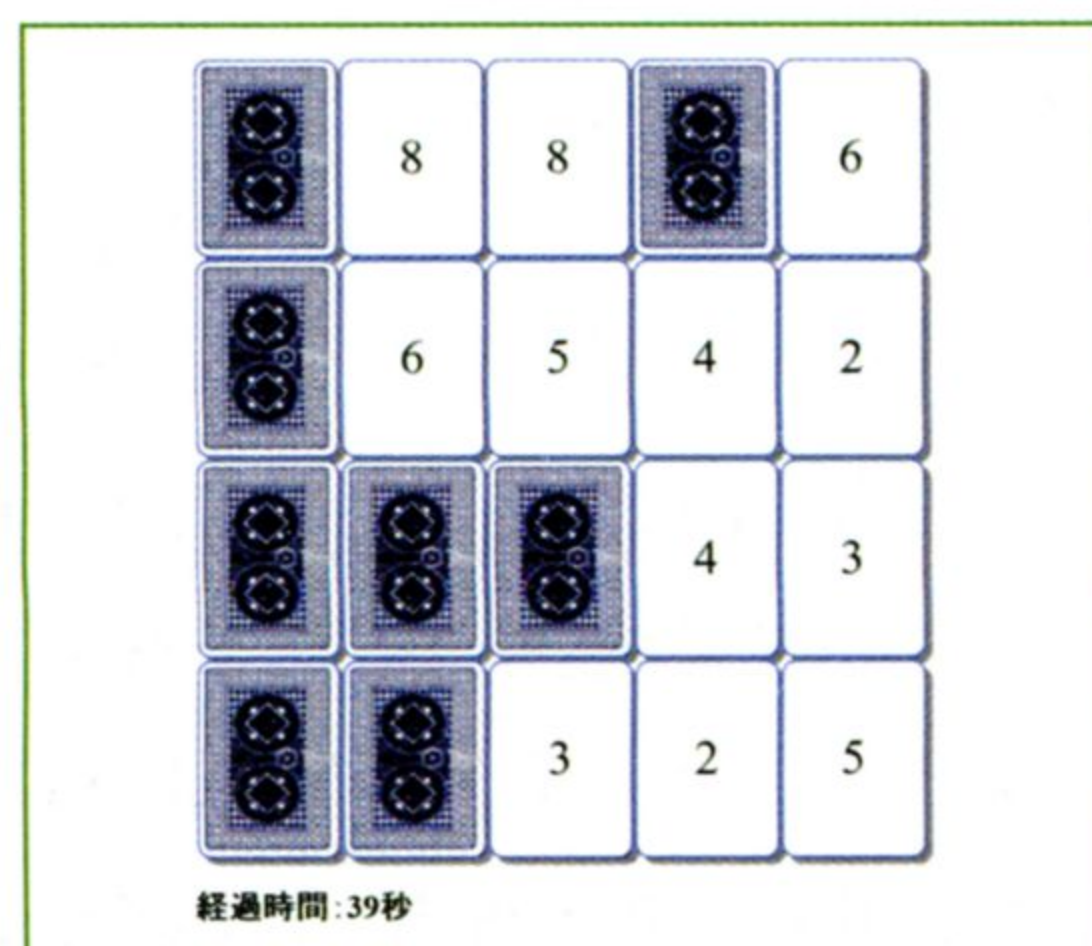
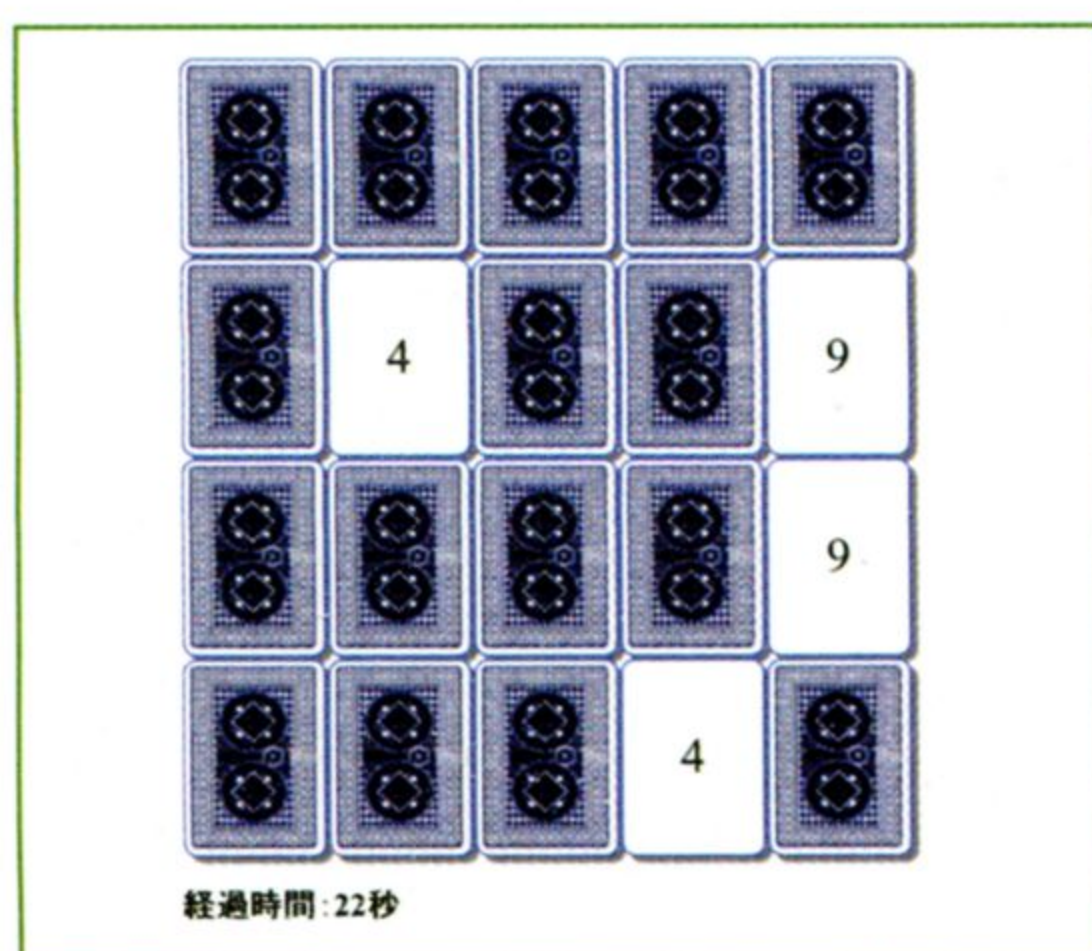
このゲームで学ぶこと

- 物理世界のオブジェクトに初速度を与える（玉の発射）

Peg

画面上部にある砲台から玉を発射します。釘にあたると効果音とともに青から赤に変わります。すべての釘の色が変わるまでの時間を競ってください。ソースは約130行です。ソースコードはほとんどPachinkoと同じです。このゲームを理解するために必要な知識は本書のほかのゲームの解説でカバーしているので、このゲームではぜひ「自分でソースコードを読んで理解する」という作業を楽しんでみてください。

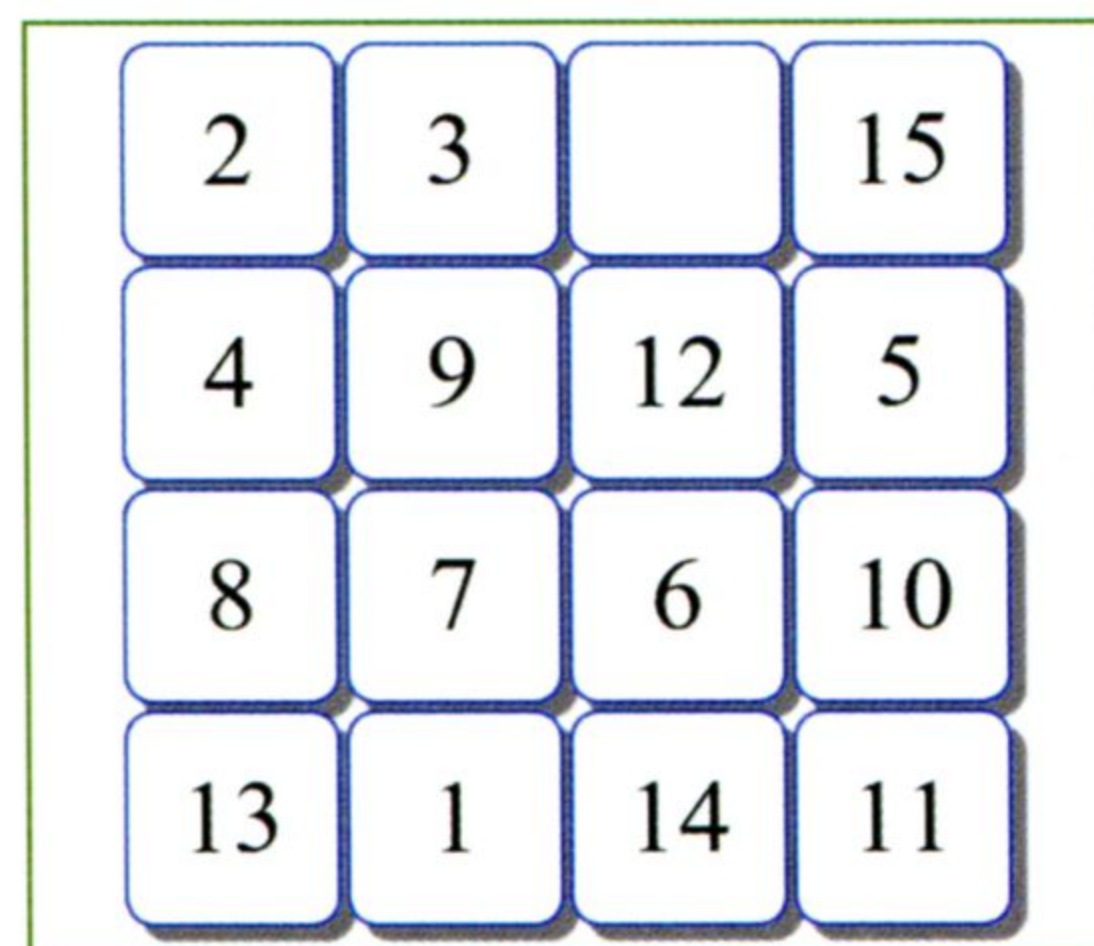
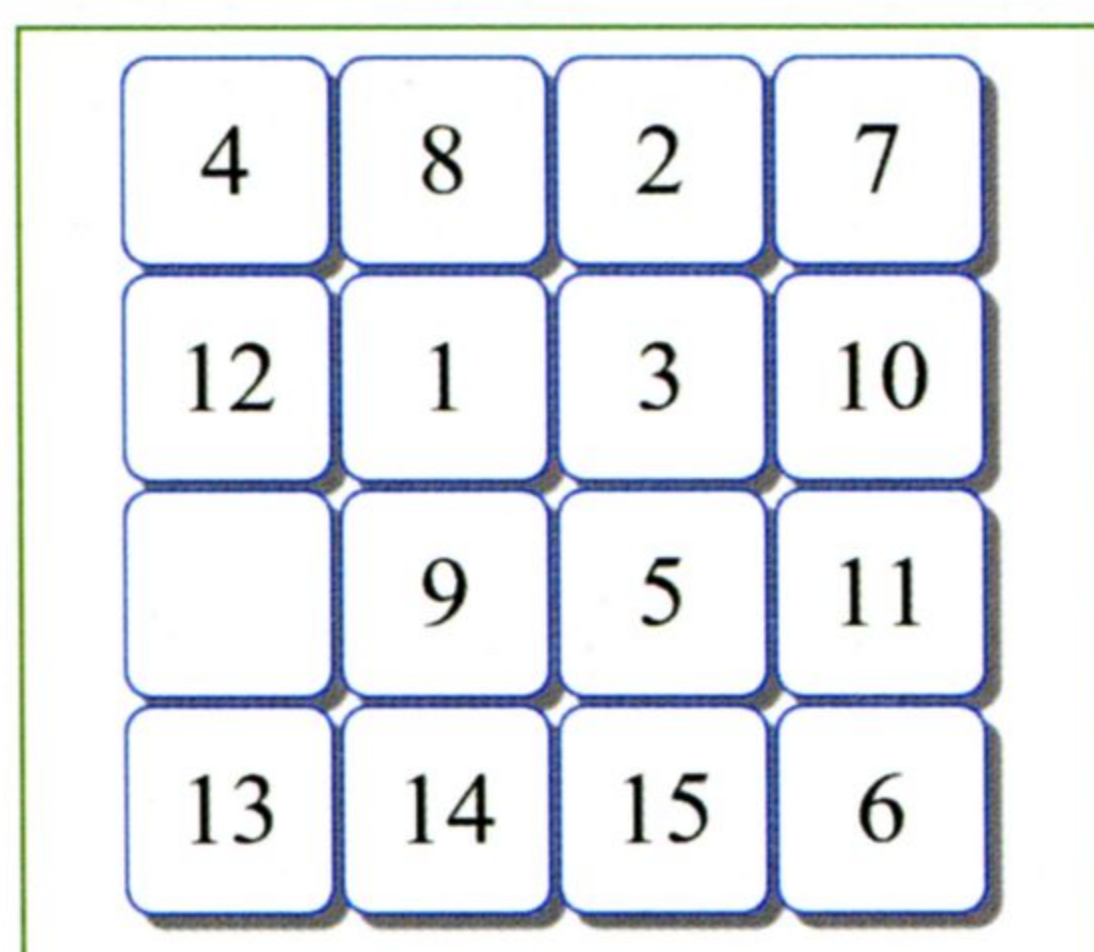
このゲームで学ぶこと

- 物理世界のオブジェクトが衝突した時に特別な処理を行う（効果音）
- クリックされた座標位置から角度を求める

◉ ◉ ◉ FlipCards

言わずと知れた神経衰弱です。すべてのカードをひっくり返すまでの時間を競います。ソースコードは約120行です。class属性をJavaScriptから操作して表示をかえる方法、配列をランダムにシャッフルする方法、数値がちがったときに裏返すタイマーの使い方などJavaScriptでの基本的な処理方法について学ぶための題材です。カードの枚数や種類を増やすなど、さまざまな工夫を凝らした神経衰弱ゲーム作成にも挑戦してみましょう。

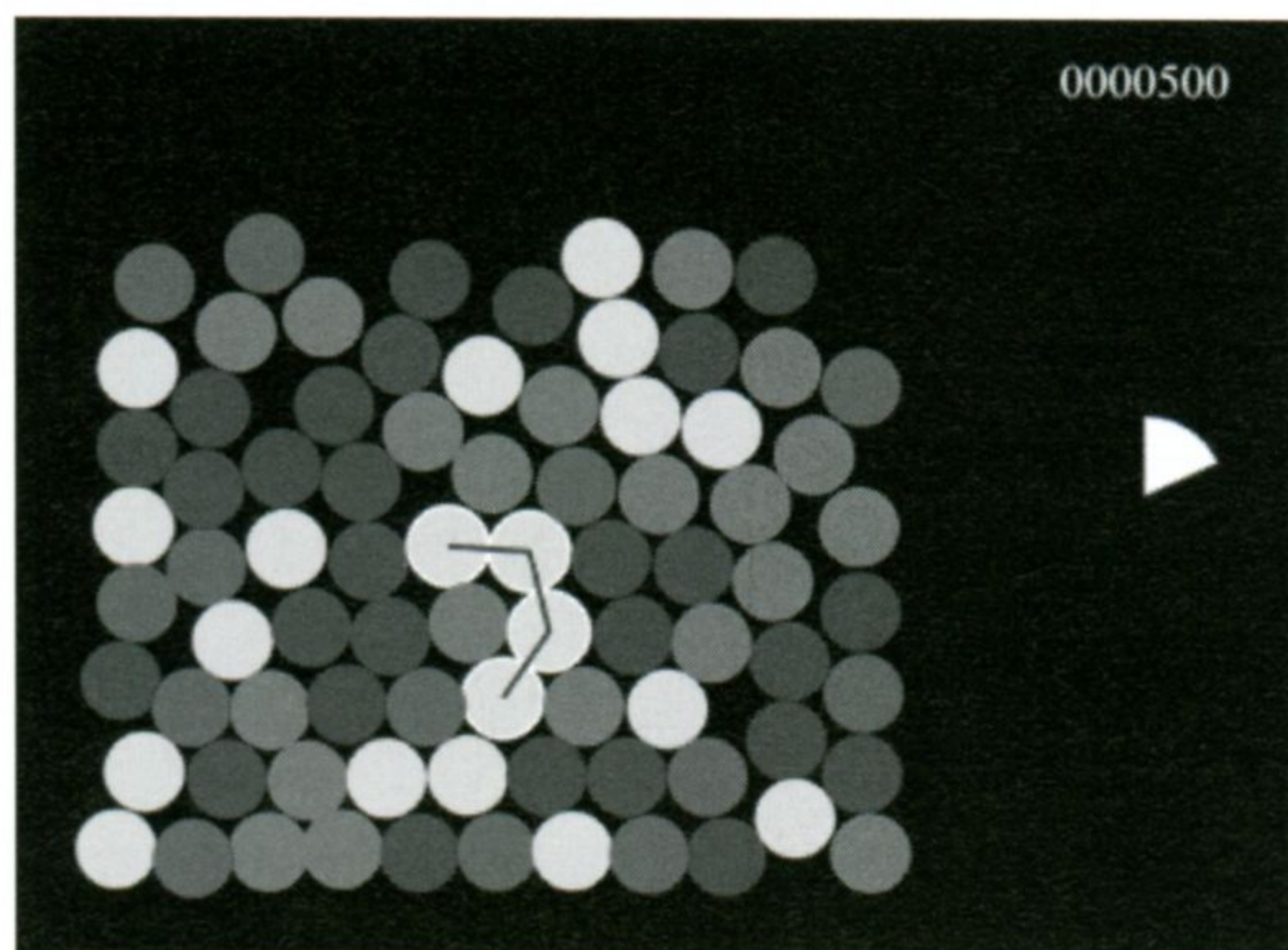
このゲームで学ぶこと

- ◉ Arrayオブジェクトのprototypeを使ってみる
- ◉ タイマーの使い方になれる

◉ ◉ ◉ 15パズル

空白タイルの上下左右をクリックすると、その数値が空白タイルと入れ替わります。その作業を繰り返して、ランダムに並んだ状態から1～15の順に整列させるというパズルです。ソースは約70行です。今回のサンプルの中では最も短いもののひとつです。JavaScriptから動的に要素を作成する方法、スタイルを使って見た目を良くする方法などの参考にしてください。また、行と列を増やすなどの変更もしてみましょう。

このゲームで学ぶこと

- ◉ DOM要素をJavaScriptから作成する
- ◉ for文の二重ループになれる

◉◉◉ デザインの重要性に関して

一般的に、開発者が単独で実装したゲームは魅力に欠けたものになりがちです。というのも、ゲームは、企画、デザイン、サウンドといったいろいろな要素が統合されて初めて面白いものになることが多いためです。今回の執筆にあたり、最初は自分でデザインを行っていました。Vegetable MarchやFunky Blocksの原型は左図のような感じでした。

前回の著書※でデザイナーの山本麻美さんと協業させていただく機会があったのですが、その際にデザインの大切さを痛感しました。自分の無味乾燥なアプリが魔法をかけられたかのように魅力的なものに変貌したのです。今回は幸いにも近所に住むデザイナーの井筒晴香さんにご協力いただくことができました。結果はご覧いただいた通りです。ゲームが見違えるようになったことに、同意いただけるかと思います。

※「JavaScriptで作るWindows ストアアプリ開発スタートガイド」(インプレス刊)

●サンプルゲームのスマホ／タブレット対応について

タブレットやスマートフォンの場合、マウスやキーボードではなくタッチを使用します。

本書でもタッチを意識してサンプルを作成しましたが、ブラウザによって挙動にばらつきがあり、すべてのブラウザに対応することは困難でした。

読者ができる限りコードを改変せずに、多くのOS/ブラウザに対応できるようにしましたが、すべてのOS/ブラウザに対応しているわけではない旨ご了承ください。

●本書で使用している物理エンジンについて

本書のサンプルゲームで使用している物理エンジンについては、インプレス社が運営するエンジニアのための技術解説サイト「Think IT」に詳細な解説記事を掲載しています。ぜひこちらもご参照ください。

[書籍連動]JavaScript物理エンジン解説

https://thinkit.co.jp/series/4770

HTML+CSSの基本

Chapter 2

HTML5
CSS
JavaScript
Canvas
Game
and
Physics engine

みなさんが目にするホームページはHTMLという言語で記述されています。この章ではHTMLの基本的な書き方とCSSについて勉強します。本章を読み終わるころには簡単なページを作成できるようになります。ぜひプログラムを入力しながら読み進んでいってください。

2-1 文書の構造

ある程度まとまった文章を書く場合、通常、見出しをつけて読みやすくするでしょう。HTMLはこのような見出しと本文といった文書の構造をWeb上で表現するためのものです。一方CSSは、見出しの書体を太くするとか、色を変えるなどの体裁を設定するためのものです。HTMLとCSSどちらもWebページをつくるには欠かせない技術です。まずはHTMLから学習していきましょう。

2-1-1 | 文書の構造

文書と言ってもいろいろですが、たとえば論文には、タイトルがあって、その下に著者名などの情報がつづき、見出し、段落が繰り返されるのが一般的です。また、手紙などでは前文、主文、末文、後付けという構成をとります。もし、順番が逆になると強い違和感を覚えるでしょう。

論文の例

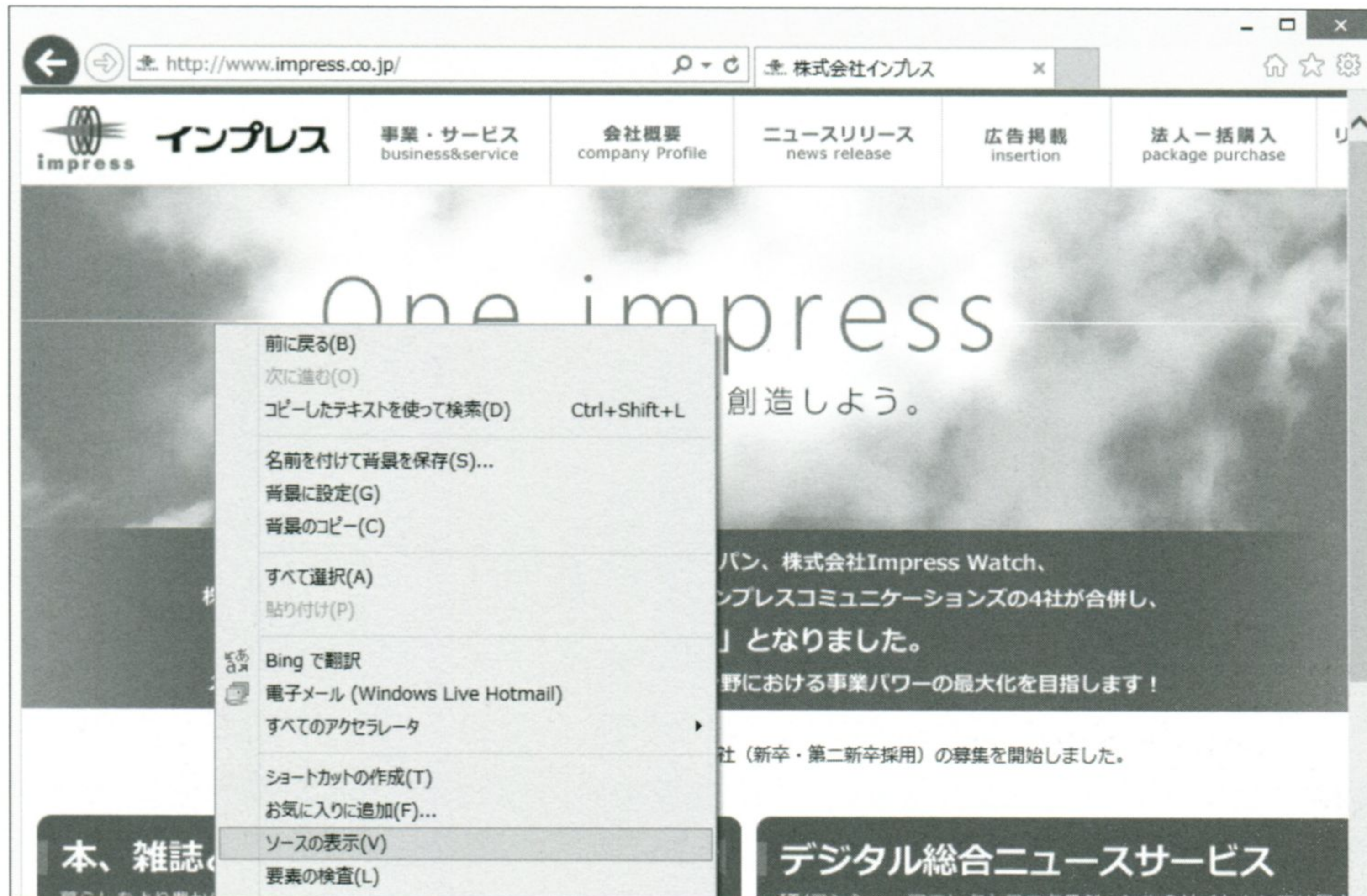
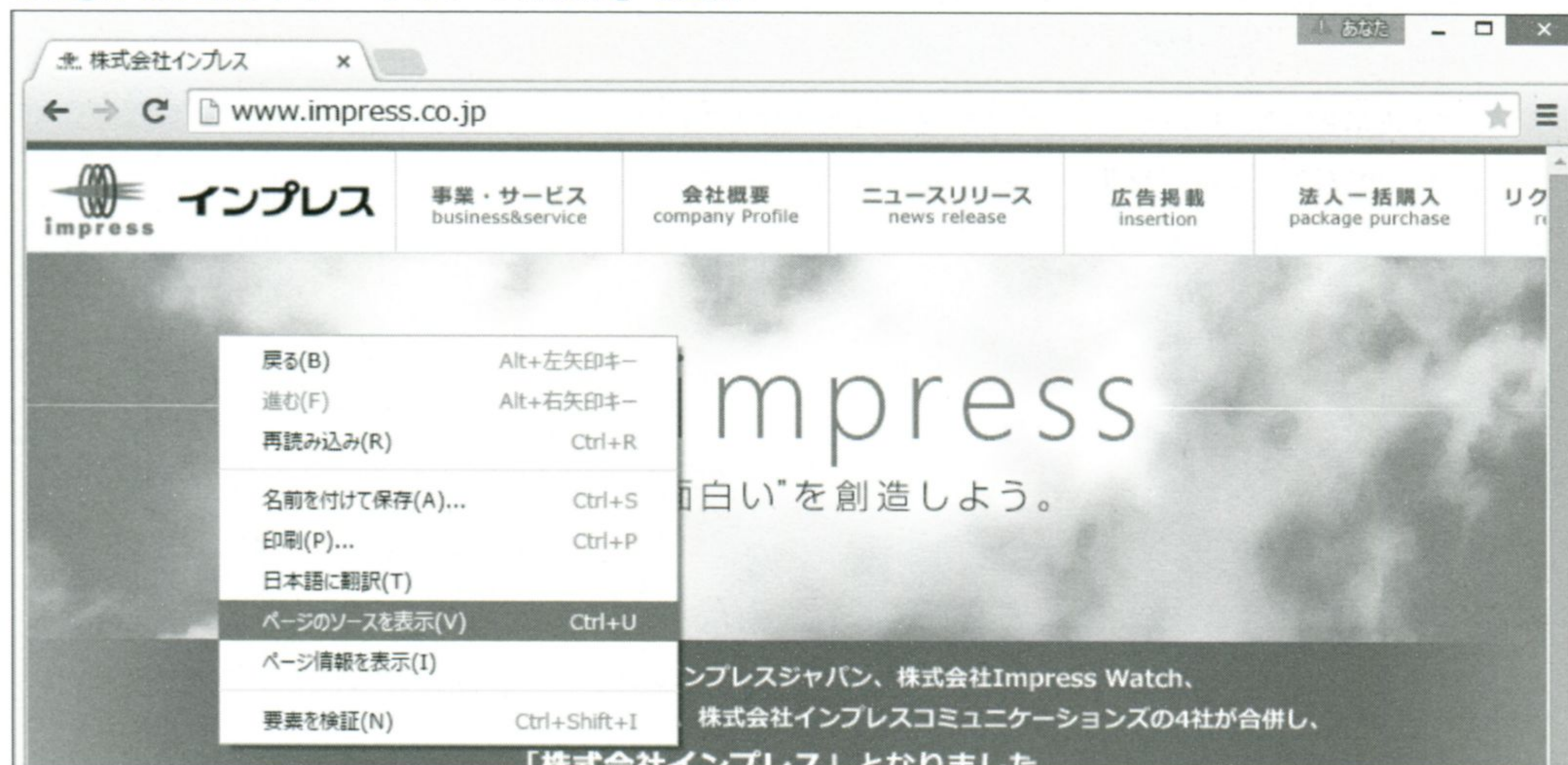
手紙の例

新聞も同じです。1面にトップニュースがあり、それぞれには大きな見出しがつけられています。写真や図が挿入されていることもあるでしょう。多くの新聞では政治面、経済面、社会面と続き、最後のページにテレビ欄という構成になっています。そのほか、取扱説明書、雑誌、小説、あらゆる文章が構造化されているといっても過言ではないでしょう。

このように、知らず知らずの間に、我々は構造化された文書に親しんでいるのです。このような文書の構造を表現しようというのがHTMLなのです。

2-1-2 ｜ 実際のページを見てみる

ではHTMLとはどのようなものでしょうか？　どんなものか見てみましょう。ブラウザでページを表示して、画面上を右クリックして「ソースの表示」(Internet Explorerの場合) という項目を選びます (Google Chromeでは「ページのソースを表示」を選びます)。ソースとはページ中身のことです。

Internet Explorerで「ソースを表示」を選択

Google Chromeで「ページのソースを表示」を選択

ブラウザによって表示状態は異なりますが、以下のようなHTMLの内容が表示されます。

表示されていたページのHTMLが表示される

http://www.impress.co.jp/ - 元のソース

ファイル(F)　編集(E)　書式(O)

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2  <html xmlns="http://www.w3.org/1999/xhtml">
3  <head>
4  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
5  <title>株式会社インプレス</title>
6  <link href="css/style.css" rel="stylesheet" type="text/css" />
7  <script type="text/javascript" src="js/jquery.js" charset="utf-8"></script>
8  <script type="text/javascript" src="js/jquery.page-scroller.js" charset="utf-8"></script>
9  <script type="text/javascript" src="newsrelease/modi.js"></script>
10 </head>
11 <body id="topidx">
12 <div id="box0">
13 <div id="menubox">
14         <ul>
15         <li><h1><a class="menu0" href="./">株式会社インプレス</a></h1></li>
16         <li><a class="menu1" href="business/">事業・サービス business&service</a></li>
17         <li><a class="menu2" href="profile/">会社概要 company profile</a></li>
18         <li><a class="menu3" href="newsrelease/">ニュースリリース news release</a></li>
19         <li><a class="menu4" href="http://ad.impress.co.jp/" target="_blank">広告掲載 insertion</a></li>
20         <li><a class="menu5" href="http://www.ips.co.jp/saiyou.shtml" target="_blank">法人一括購入 package purchase</a></li>
21         <li><a class="menu6" href="recruit/">リクルート情報 recruitment</a></li>
22     </ul>
23 </div>
24 <a id="top"></a>
25 <div id="mainbox">
26     <h2>meinvisual</h2>
27     <div id="topinfo">
28
29     ▶ <span style="color: #0e91ce;">お知らせ（2015.05.21）</span> <a href="http://job.mynavi.jp/16/pc/search/corp79615/outline.html"
   target="_blank">2016春入社（新卒・第二新卒採用）の募集を開始しました。</a><br />
30     </div>
31
32     <div id="topbox">
33         <div id="topbox1"><h4><a href="business/index.html#business1">本、雑誌と関連 Web サービス</a></h4>
34       <p class="txt"><a href="business/index.html#business1"><img src="img/txt_topbox1.png" alt="暮らしをより豊かに楽しくする、IT、PC、デジ
   タル製品をテーマに、関連する技術情報から利用方法まで幅広く提供する出版&Webサービス事業を展開しています。" width="440" height="60"
   border="0" /></a></p><p class="partner"><img src="img/img_topbox1.png" alt="メディア&サービス" width="440" height="135" border="0"
   usemap="#Map" />
```

一見すると難しそうに見えますが、基本となるルールは実はとてもシンプルです。これからそのルールについて見ていきましょう。

演習　いろいろなページのソースを見てみよう

2-2 最初のHTML

本節から実際のHTMLに触れていきます。単に読むのと手を動かしながら読むのでは理解の度合いが格段に違ってきます。ぜひ、自分で入力して結果を確認しながら読み進んでください。

2-2-1 | HTMLの「< >」記号に注目

多くの文書はなんらかの構造を持っています。ホームページも例外ではありません。では、「見出しはここからここまで」「段落はここからここまで」ということを、どのように記述するのでしょうか? HTMLのソースを見たときに「<」や「>」といった記号がたくさんあったことに気づいたと思います。実は、この「<」と「>」が文書を構造化するための印、すなわち「見出しの範囲」や「段落の範囲」を示す印だったのです。「<」と「>」に囲まれた部分が開始で、「<」と「/>」に囲まれたのが終了の印です。具体的には以下のようになります。

<見出し>見出しの内容</見出し>

<段落>段落１の内容</段落>

<段落>段落２の内容</段落>

ただ、HTMLは、英語圏で仕様が策定されたこともあり、構造の内容はすべて英単語で記述します。たとえば、文書のタイトルは<title>、文書の内容は<body>、見出しは<h1>、<h2>、<h3>、段落は<p>のように、それぞれの用途が定められています。

実際のHTML文書を見てみましょう。

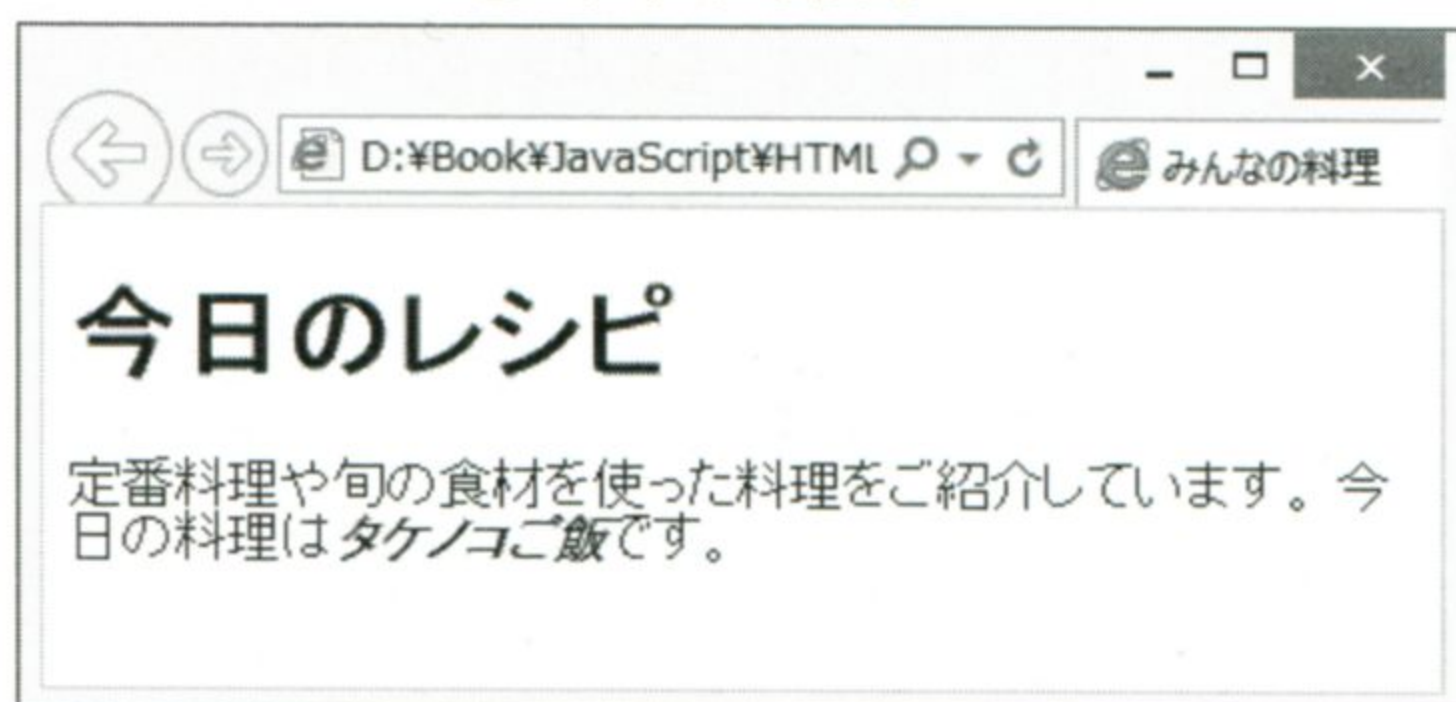
HTML文書の例 SAMPLE html-basic1.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>みんなの料理</title>
    </head>
    <body>
        <h1>今日のレシピ</h1>
        <p>
            定番料理や旬の食材を使った料理をご紹介しています。
            今日の料理は<i>タケノコご飯</i>です。
        </p>
    </body>
</html>
```

前ページHTMLをブラウザで表示

D:¥Book¥JavaScript¥HTML　みんなの料理

今日のレシピ

定番料理や旬の食材を使った料理をご紹介しています。今日の料理は*タケノコご飯*です。

「<」と「>」に囲まれた部分を「タグ」もしくは「要素」と呼びます。ちなみに英語ではtagあるいはelementと呼びますが、辞書でtagを調べると「区別する、タグを付ける、認識する」などと説明されています。つまり、<開始タグ>と</終了タグ>で囲むことで、その部分がどのような意味を持つのか、すなわちどんな構造なのかを表現しているのです。

たとえば、ある部分を斜体（italic）で表現したい場合は、「今日の料理は<i>タケノコご飯</i>です」のように記述します。こうすると「タケノコご飯」の部分だけが斜体で表示されます。

ちなみに、HTMLはHyper Text Markup Languageという意味ですが、このMarkupとは「印をつける」という意味です。タグで囲んで文章の構造を表現するのでMarkup Languageと呼ばれているのです。

演習　前ページのHTMLを入力して表示してみよう

❶ 「メモ帳」などのエディタを開く（入力ツールについてはP.038「1-5 統合開発環境のすすめ」参照）。
Windowsキーを押して「note」と入力するとメモ帳が検索されるので、それをクリックするとメモ帳が起動します。
Macでは「テキストエディット」などを利用できます。

❷ HTMLコードを入力。
誤字がないよう正確に入力しましょう。

❸ 「Sample1.html」として保存。
ファイルの種類は書式情報のないプレーンなテキスト形式にします。保存時のダイアログで「テキスト文書」（アプリによって「テキストドキュメント」「テキスト形式」「標準テキスト」「プレインテキスト」など）を選択してください。Macのテキストエディットの場合は、「フォーマット」メニューから「標準テキストにする」を選択します。

❹ そのファイルをダブルクリックして表示する。
ダブルクリックしてもブラウザが立ち上がらない場合は、ファイルの拡張子が正しく設定されているか確認してください。

▶文字コードについて

<html>の直後の<head>要素の中に、<meta charset="UTF-8">とあるのは、「この文書の文字はUTF-8という形式で保存されていますよ」ということを宣言するためのものです。

昔は今ほど標準化が進んでいなかったため、S-JIS、EUCなどのエンコーディング方式が混在して使われてきました。エンコーディングとは文字をどのように数値に変換するかというルールのことです。たとえば「あ」という文字を保存する場合、EUCではA4A2として、S-JISでは82A0として保存されます。同じ文字を保存するのにまったく異なる情報が保存されるのです。

仮にEUCで保存したファイルをS-JISとして読もうとするとグチャグチャな文字になってしまいます。「ホームページで文字が化けて読めない!」という経験をした読者の方もいるかもしれません。これはページ作成者がファイルを保存したときに使用した文字エンコーディングをブラウザー側で正しく認識できなかったことが原因です。

最近ではUTF-8が主流となってきました。みなさんが自分でHTMLを記述する際も、UTF-8形式で保存し、<meta>タグを挿入する習慣をつけるとよいでしょう。本書のサンプルには<meta charset="UTF-8">がないものがあるかもしれませんが、コンテンツを作成するときには挿入するようにしたほうがよいでしょう。

ファイルの拡張子

ファイルの拡張子とはファイル名の後ろに付与される短い文字列のことです。たとえば、「Sample1.html」というファイル名の場合、「.html」が拡張子に相当します。パソコンはこの拡張子とアプリケーションを関連づけて管理しています。たとえば、「.htmlという拡張子を持つファイルはChromeで開く」という感じです。ダブルクリックしてもブラウザが立ち上がらない場合は、ファイルの拡張子とアプリケーションが正しく関連付けられているか確認してください。

2-3 HTMLの書き方の規則

HTMLの書き方には規則がありますが、難しいものではありません。これまで見てきたように、基本は文字列（文字の並び）を「<要素名>文字列</要素名>」で囲みます。要素名を使って、見出しや段落など文書の構造を表します。

2-3-1 タグの書き方

まず、タグの書き方の基本ルールをおさえておきましょう。

ルール1：タグには開始タグと終了タグがあり、その中に中身を記述する

開始タグは「<」記号と「>」記号で囲み、終了タグは「</」記号と「>」記号で囲みます。

タグの書き方

```
<要素名>□□□中身□□□</要素名>
↑                    ↑
開始タグ              終了タグ
```

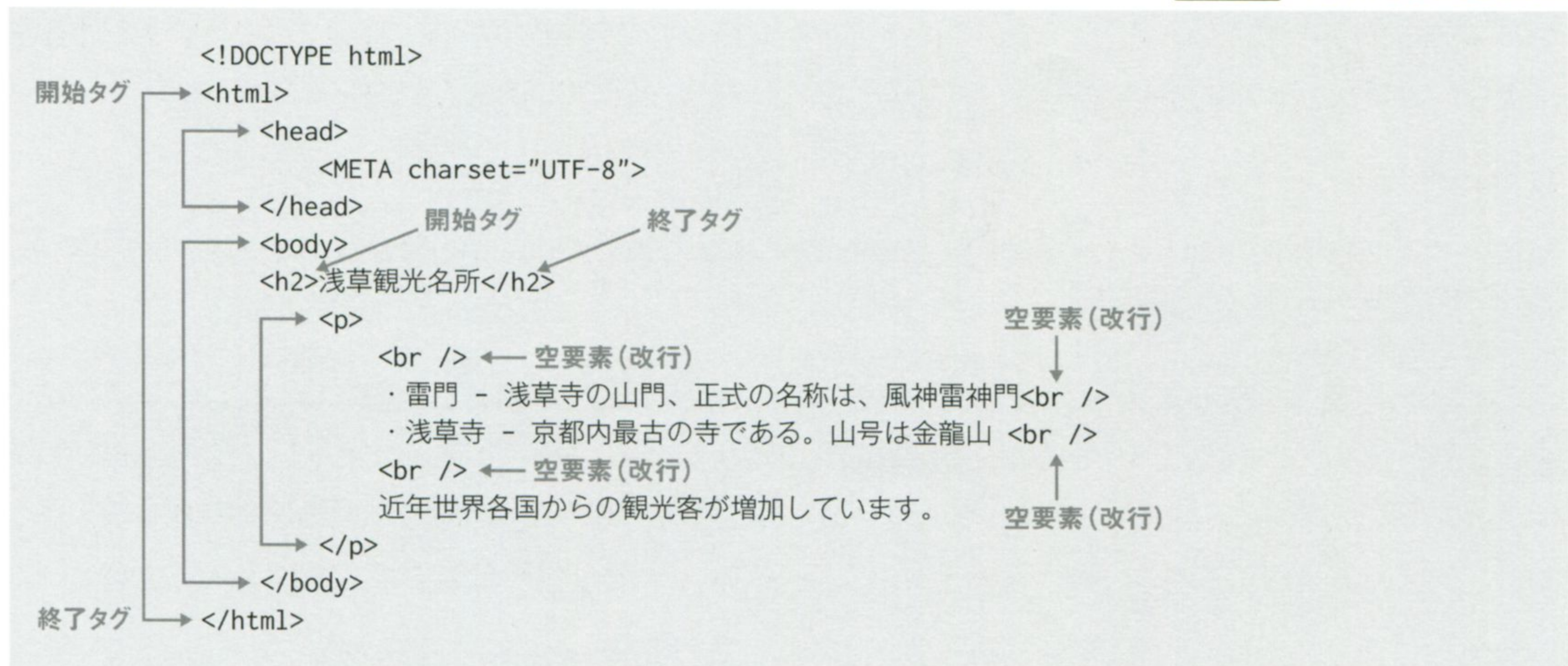
ルール2：中身がない要素は「空要素」と呼び、以下のように記述する

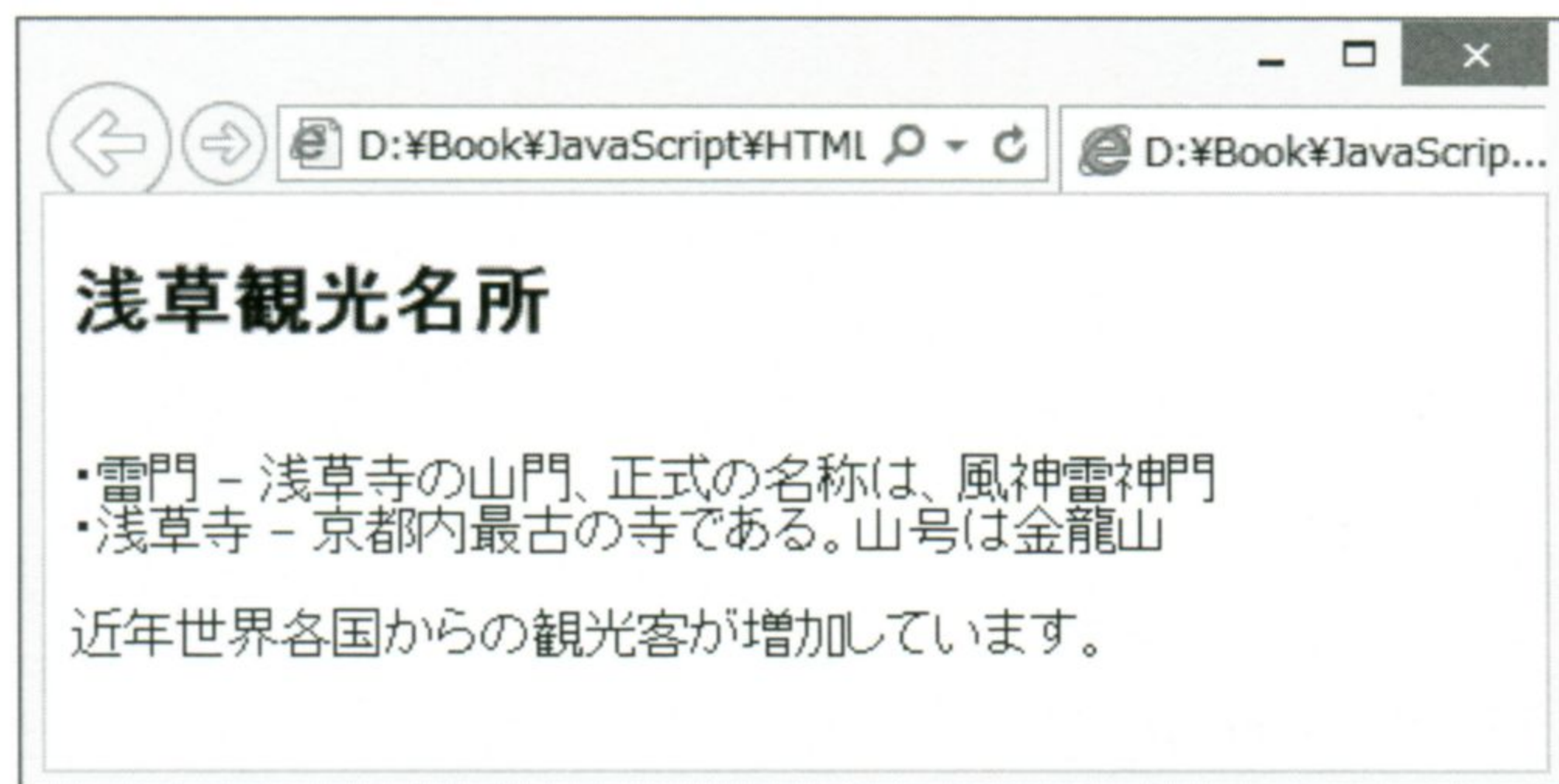
代表的なものに改行する
や、画像を表示する<img />があります。

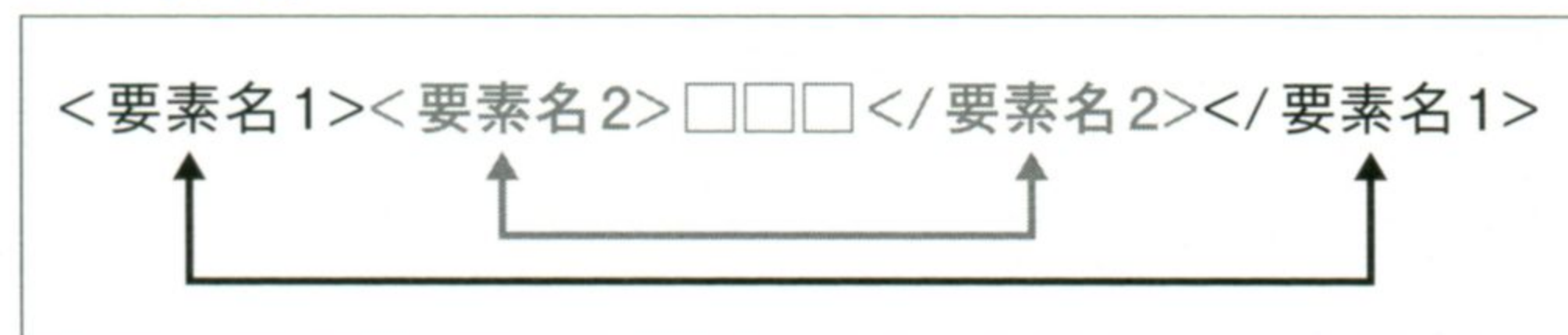
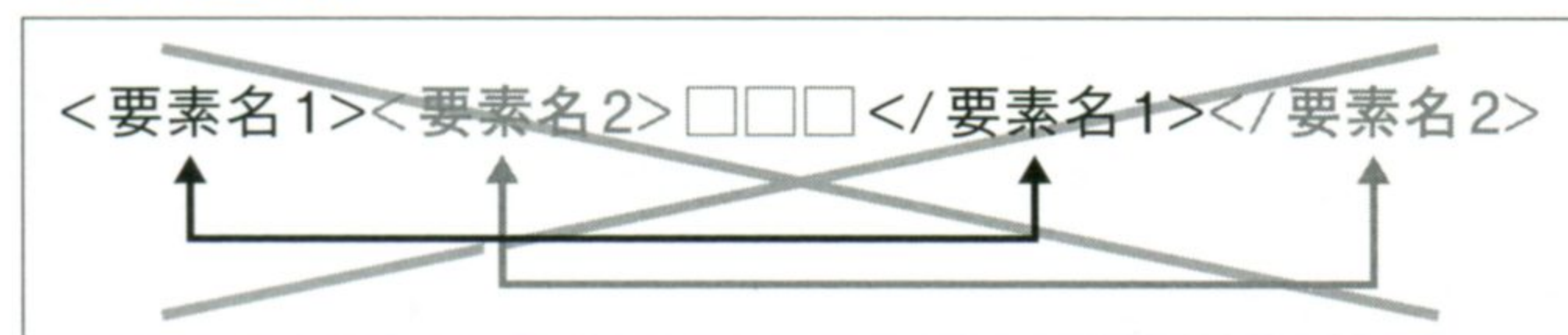
空要素の書き方

```
<要素名 />
```

HTMLの書き方の例　SAMPLE html-basic2.html

```
          <!DOCTYPE html>
開始タグ → <html>
            <head>
                <META charset="UTF-8">
            </head>
            <body>        開始タグ        終了タグ
            <h2>浅草観光名所</h2>
                <p>                                      空要素(改行)
                    <br /> ← 空要素(改行)
                    ・雷門 - 浅草寺の山門、正式の名称は、風神雷神門<br />
                    ・浅草寺 - 京都内最古の寺である。山号は金龍山 <br />
                    <br /> ← 空要素(改行)
                    近年世界各国からの観光客が増加しています。    空要素(改行)
                </p>
            </body>
終了タグ → </html>
```

NOTE <html>の上にある<!DOCTYPE html>は「このHTML文書はHTML5を意識してつくられたものです」と宣言するためのものです。ブラウザはこの情報を見て「この文書はHTML5を想定してつくられたんだな」と認識することができます。この行はHTMLの先頭に記述されますが、画面に表示されることはありません。「おまじない」のようなものだと思っておいてください。

前ページのHTMLをブラウザで表示

ルール3：タグは別のタグを含むことができますが、その場合は全体を含んで入れ子にします。

正しい囲み方

間違った囲み方

たとえば、以下の「正しい例」では、内側のタグ<i>の範囲が外側のタグ<p>に含まれているので問題ありません。しかし、「間違っている例」はそうなっていないので不正なHTMLとなります。

正しい例

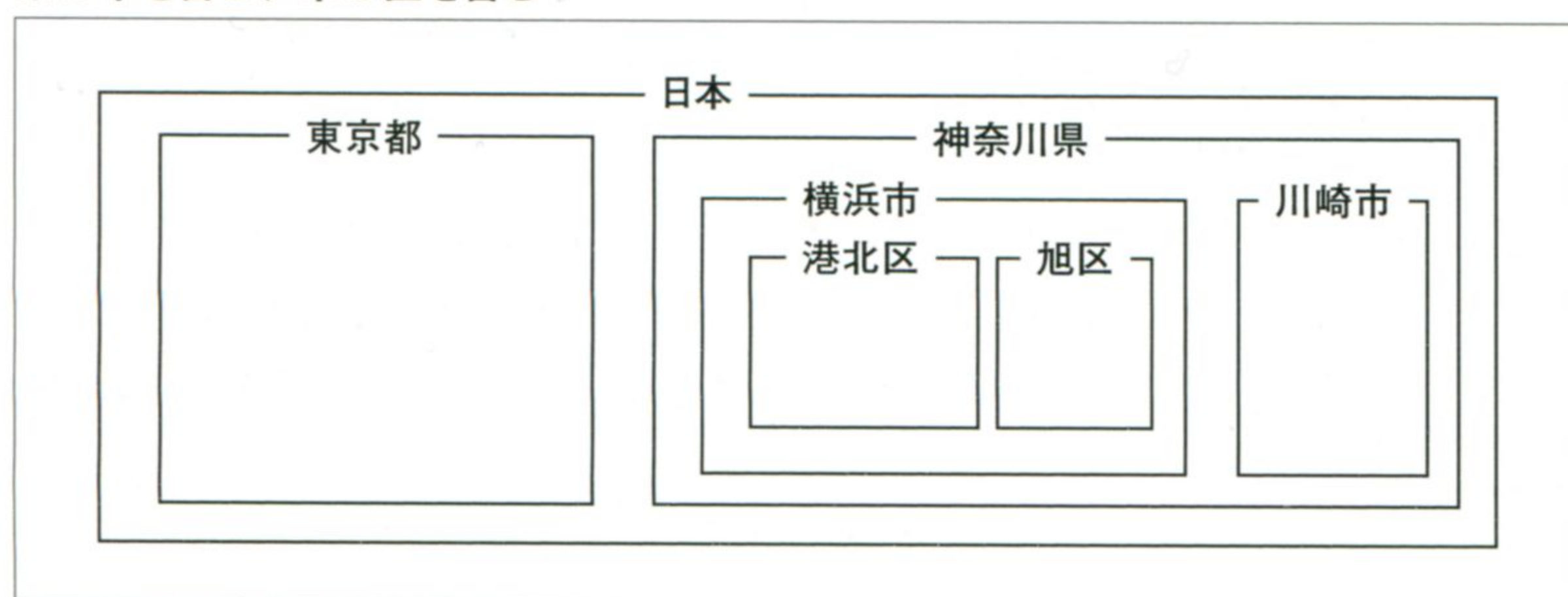
```
<p>今日の天気は<i>快晴</i>です</p>
```

間違っている例

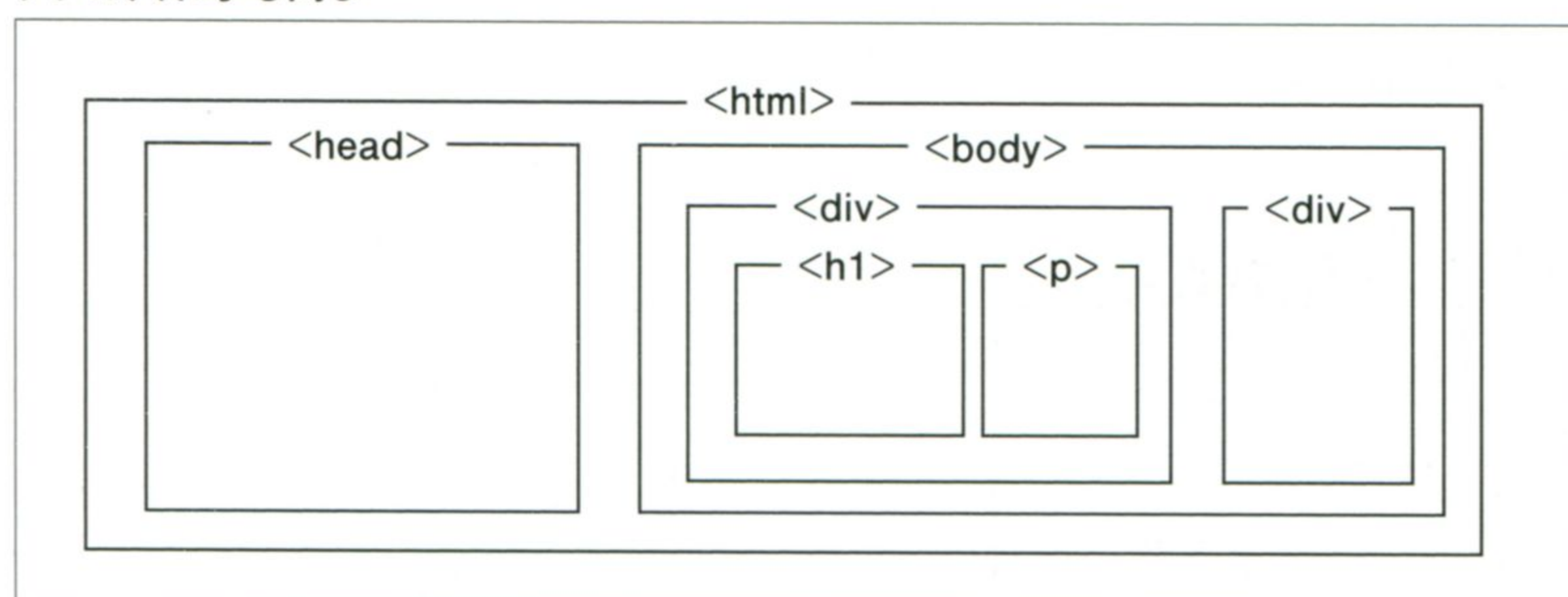
```
<p>今日の天気は<i>快晴</p>です</i>
```

ちょっとわかりにくいかもしれないので例を使って説明しましょう。たとえば、神奈川県には横浜、川崎といった市がありますが、これらの市が東京都にはみ出すことはありません。さらに、横浜市は中区、旭区、港北区、神奈川区といった区を含んでいますが、これらの区の一部が川崎市に含まれることはありません。HTMLでは、含む方を「親」、含まれる方を「子」と表現しますが、子は必ず親に完全に含まれる必要があるのです。

県は市を含む、市は区を含む

日本
東京都
神奈川県
横浜市
港北区
旭区
川崎市

上図をHTML文書に置き換えてみましょう。

タグの入れ子も同じ

<html>
<head>
<body>
<div>
<h1>
<p>
<div>

すべての要素は親要素に完全に含まれている様子がわかると思います。仮にはみ出してしまった場合、どのように処理されるかはブラウザによって異なります。

ルール4：一番外側の要素は<html>がひとつだけ

ルール5：要素に付属情報が必要な場合は属性として記述します。属性は複数指定することもできます。

```
<要素名 属性名1="属性値1" 属性名2="属性値2">
```

例）id属性がtitleという値で、class属性がblueという値を持つp要素

```
<p id="title" class="blue"> .... </p>
```

ちなみにid属性は、HTMLの文書中で要素を特定するために使用します。パスポートや運転免許所で個人を特定するようなイメージです。一方、class属性は、いくつかの要素をまとめて処理するときに使用します。赤色の服を着ている人々、年齢20歳代の人々、というように対象となる人々を選択するようなイメージです。id属性もclass属性も、今後たくさん例が出てきますので、ここでは「ふぅ〜ん、そんなものがあるのか…」程度の認識でかまいません。

主なルールはこれくらいです。一見すると非常に複雑に見えるHTMLですが、ルールは非常にシンプルだということがおわかりいただけたと思います。

2-4 HTMLの主な要素

HTML辞典などの書籍や、要素を解説しているWebサイトなどには、膨大な数の要素があり、圧倒されるかもしれませんが、それらを全部覚える必要はありません。代表的な要素をいくつか覚えれば十分です。必要に応じて徐々に引き出しを増やしていきましょう。

2-4-1 これだけは覚えておきたい必須要素

よく使用される要素とその使用例をご紹介します。

覚えておきたい要素

要素名	元になった英語	用途
h1	heading 1	見出し1
h2	heading 2	見出し2
h3	heading 3	見出し3
p	paragraph	段落
div	document division	グループ化
span	span	テキストの一部
ul	unordered list	箇条書き
ol	ordered list	番号付リスト
li	list item	リストの項目
img	image	画像
br	break	改行
button	button	ボタン
input	input	インプット
a	anchor	文書間のリンクを記述
table	table	テーブル
tr	table row	テーブルの行
td	table data	テーブルのセル

NOTE 昔はいろいろな要素を駆使して、ページの見映えを競う傾向がありましたが、最近はCSS（P.040「2-6 CSSの概要」）を使って調整する方法が主流になってきました。

▶見出しと段落に使う要素 — h1、h2、h3、p、span

<h1>、<h2>、<h3>はそれぞれ見出しです。<p>は段落を示します。<span>は文章の一部に特別な意味を与える役目をしますが、ブラウザでの表示は周囲の文字と同じです。<span>要素は、CSSを使ってその部分だけ表示を変えたり、JavaScriptから値を書き換えたりするときに便利に利用できます。また、複数の要素をひとつにまとめる<div>要素も実際のコンテンツで多用されます。

使用例 SAMPLE html-major-element1.html

```
<!DOCTYPE html>
<html>
    <head>
        <META charset="UTF-8">
    </head>
    <body>
        <h1>これは見出し１です</h1>
        <h2>これは見出し２です</h2>
        <h3>これは見出し３です</h3>
        <p>
            これは段落です。上と下に空白が挿入され、まとまった塊として提示されます。
        </p>
        <p>
            これも段落です。
            span要素を使うと<span>文章の一部</span>に特別な意味を持たせることができます。
        </p>
    </body>
</html>
```

ブラウザ表示結果

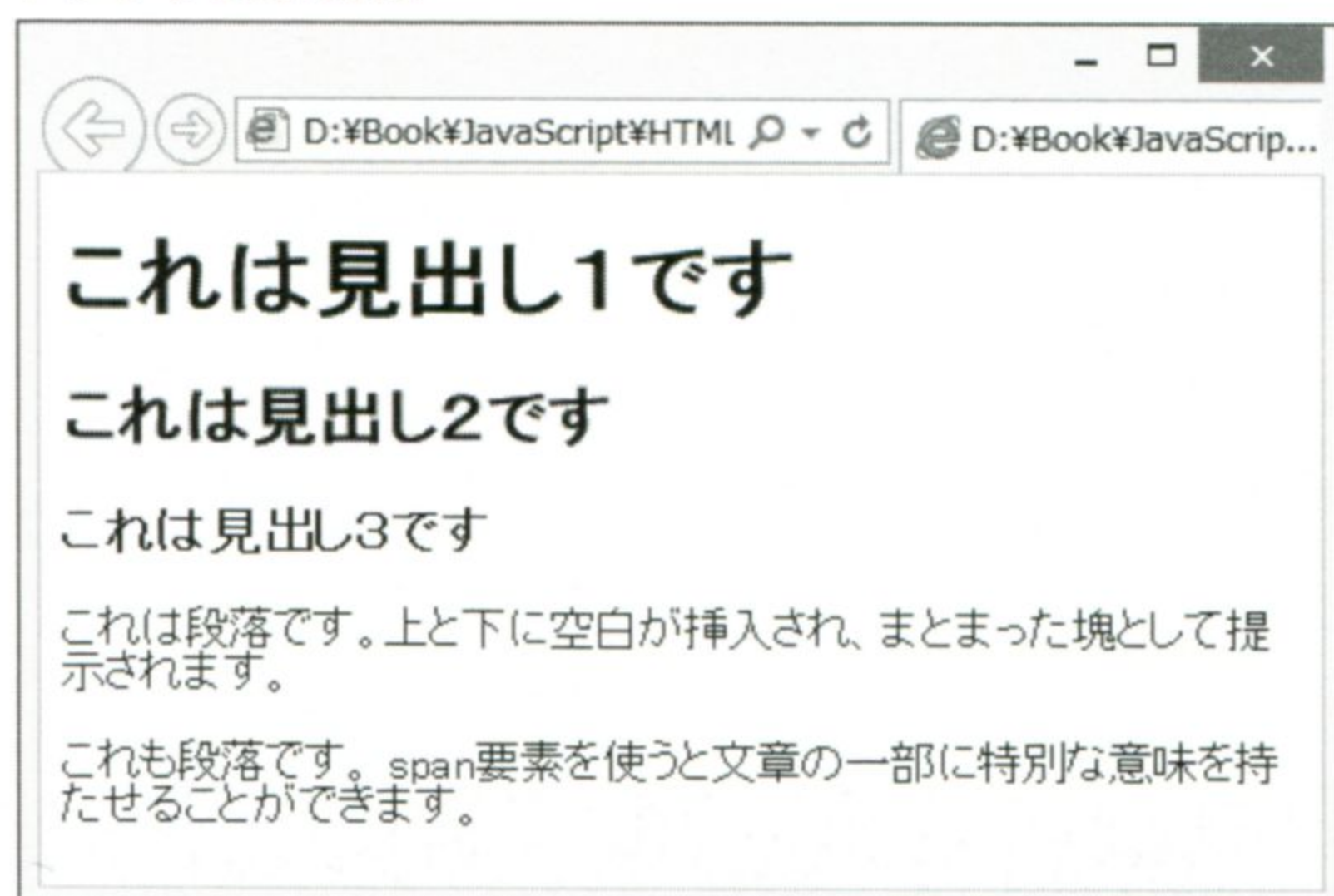
D:¥Book¥JavaScript¥HTML
D:¥Book¥JavaScrip...

これは見出し1です

これは見出し2です

これは見出し3です

これは段落です。上と下に空白が挿入され、まとまった塊として提示されます。

これも段落です。span要素を使うと文章の一部に特別な意味を持たせることができます。

▶箇条書き — ol、ul、li

は箇条書き（番号なしリスト）、<ol>は番号付のリストです。<li>はリストの項目です。箇条書きはさまざまな文書で利用されますが、これも立派な構造のひとつです。

使用例 SAMPLE html-major-element2.html

```
<!DOCTYPE html>
<html>
    <head>
        <META charset="UTF-8">
    </head>
    <body>
        <h2>ジャーマンポテト</h2>
        <h3>材料</h3>
        <ul>
            <li>ジャガイモ　2個</li>
            <li>玉ねぎ　1個</li>
            <li>ベーコン　200g</li>
        </ul>
        <h3>作り方</h3>
        <ol>
            <li>ジャガイモと玉ねぎを薄切りにします</li>
            <li>ベーコン、ジャガイモ、玉ねぎの順で炒めます</li>
            <li>塩コショウで味を調えます</li>
        </ol>
    </body>
</html>
```

ブラウザ表示結果

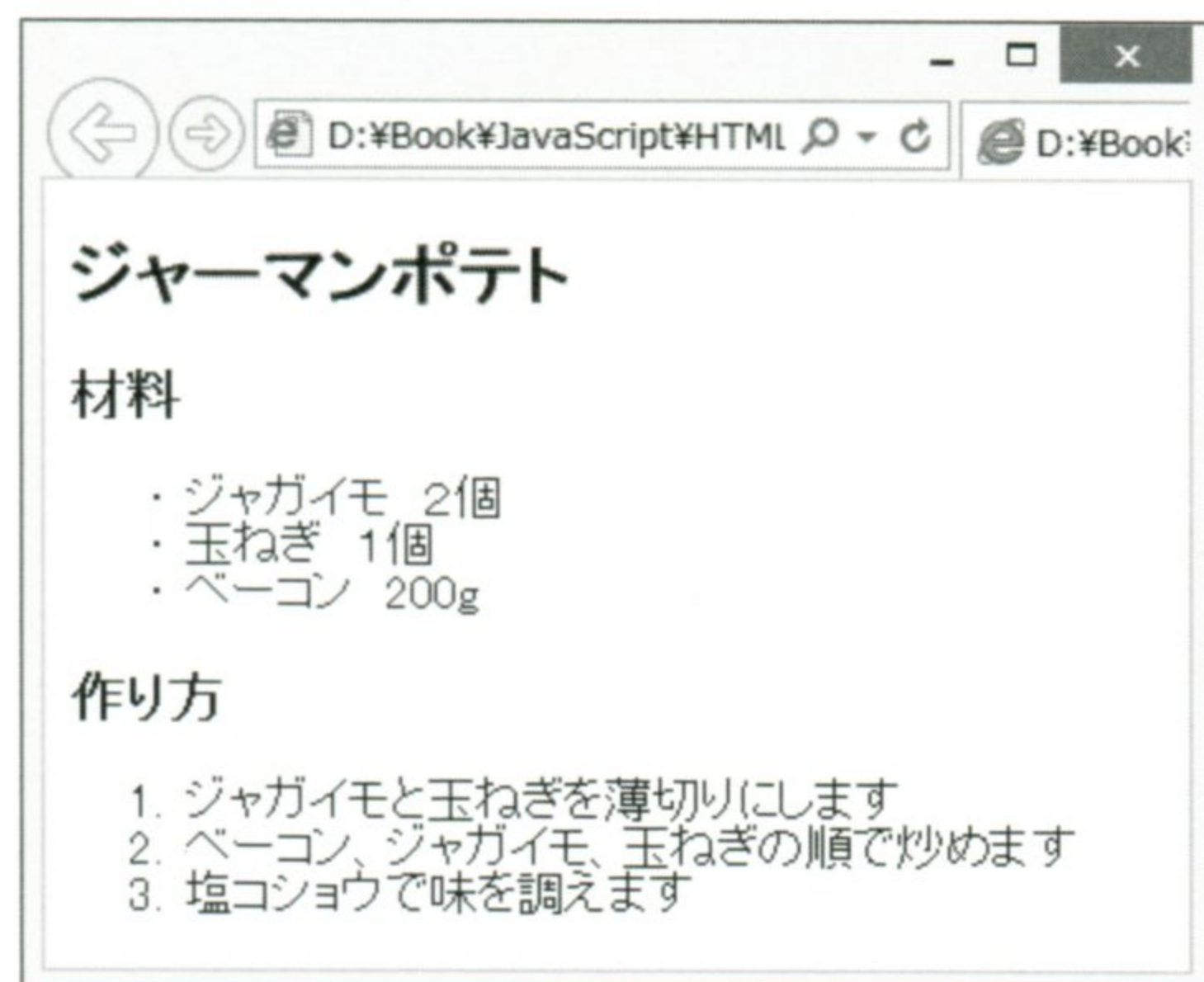
D:¥Book¥JavaScript¥HTML　D:¥Book

ジャーマンポテト

材料

・ジャガイモ　2個
・玉ねぎ　1個
・ベーコン　200g

作り方

1. ジャガイモと玉ねぎを薄切りにします
2. ベーコン、ジャガイモ、玉ねぎの順で炒めます
3. 塩コショウで味を調えます

▶テーブル ― table、tr、td

「時間割」「野球のスコア」「テレビ番組欄」「カレンダー」「オセロや将棋盤」あらゆるところに表形式の表現が使われています。<table>はこのような構造に使用します。使い方は以下のとおりです。

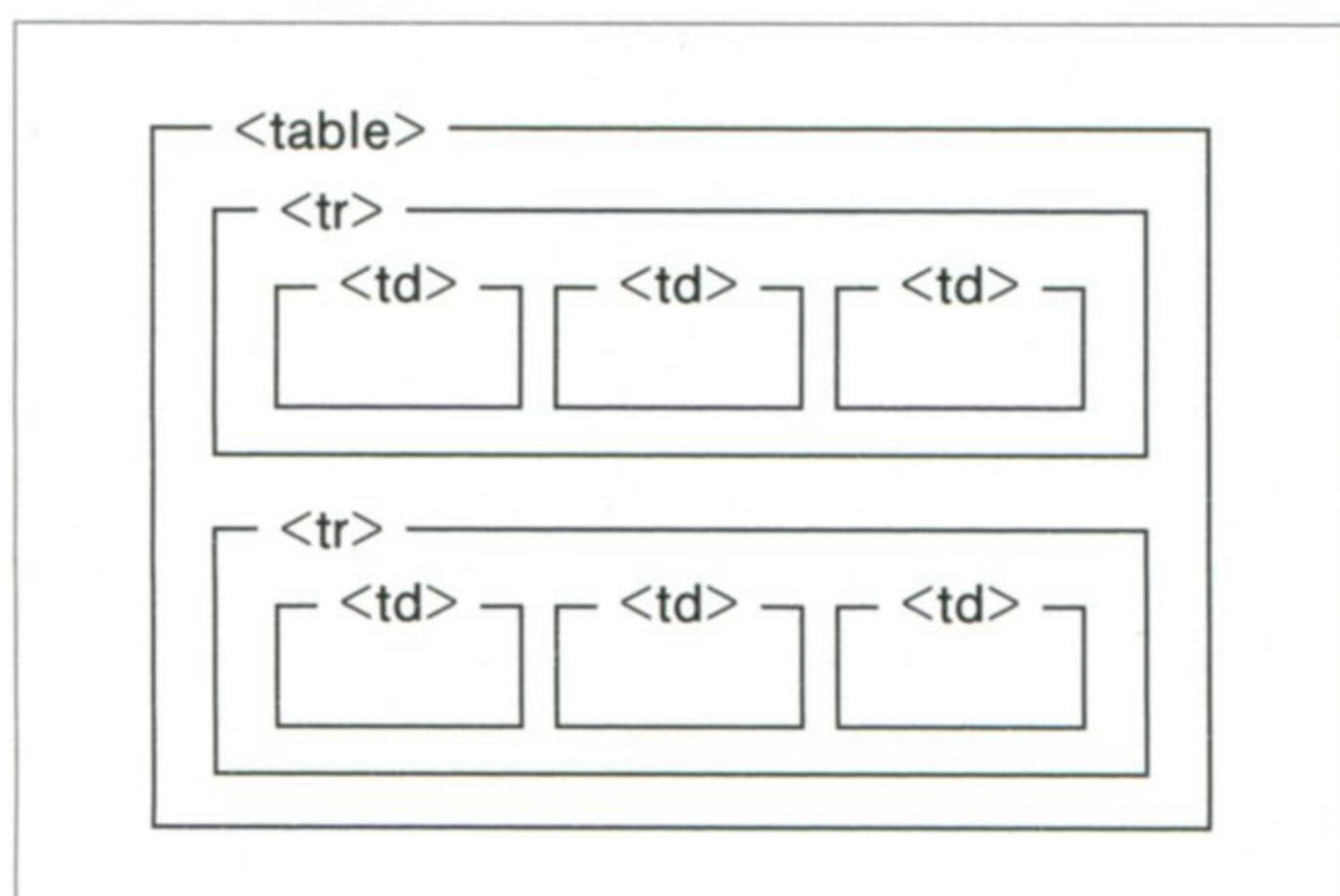
- 表の全体を<table border="1">要素で表現する。枠の太さはborder属性で指定する
- 各行（横方向）を<tr>要素で表現する
- 行の中の個々の要素を<td>で表現し、その中に表示する内容（文字や図）を配置する

テーブルの構造

<table>
<tr>
<td> <td> <td>
<tr>
<td> <td> <td>

行（横方向）を最初につくって、その中に縦方向のマスを配置していくのがポイントです。列（縦方向）を最初につくることはできません。

使用例 SAMPLE html-major-element3.html

```
<!DOCTYPE html>
<html>
    <body>
        <h2>今日の試合</h2>
        <table border="1">
            <tr>
                <td></td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>計</td>
            </tr>
            <tr>
                <td>横浜</td><td>1</td><td>0</td><td>0</td><td>2</td><td>1</td><td>4</td>
            </tr>
            <tr>
                <td>阪神</td><td>1</td><td>3</td><td>0</td><td>0</td><td>0</td><td>4</td>
            </tr>
        </table>
    </body>
</html>
```

ブラウザ表示結果

D:¥Book¥JavaScript¥HT

今日の試合

	1	2	3	4	5	計
横浜	1	0	0	2	1	4
阪神	1	3	0	0	0	4

横方向にマスを連結する場合はcolspan属性を、縦方向に連結する場合はrowspan属性を使用します。連結するマスの数を属性値として指定します。縦・横ともに<td>要素の属性であることに注意してください。

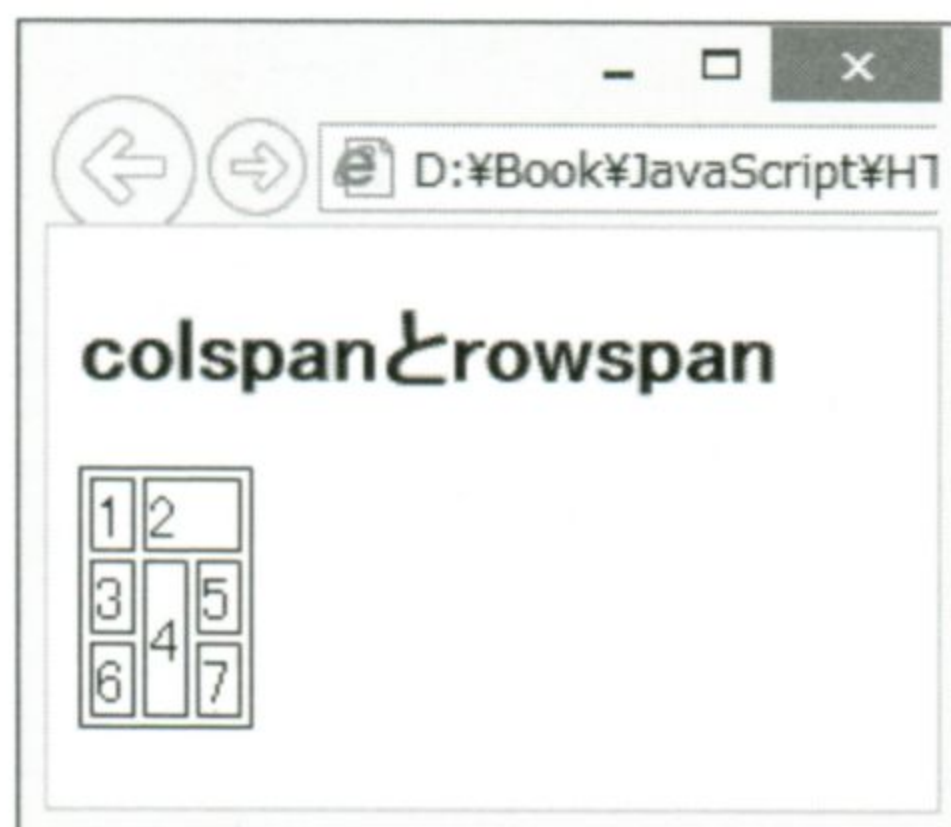
使用例　**SAMPLE** html-major-element4.html

```
<!DOCTYPE html>
<html>
    <head>
        <META charset="UTF-8">
    </head>
    <body>
        <h2>colspanとrowspan</h2>
        <table border="1">
            <tr>
                <td>1</td>
                <td colspan="2">2</td>
            </tr>
            <tr>
                <td>3</td>
                <td rowspan="2">4</td>
                <td>5</td>
            </tr>
            <tr>
                <td>6</td>
                <td>7</td>
            </tr>
        </table>
    </body>
</html>
```

ブラウザ表示結果

D:¥Book¥JavaScript¥HT

colspanとrowspan

<table border="1"><tr><td>1</td><td colspan="2">2</td></tr><tr><td>3</td><td rowspan="2">4</td><td>5</td></tr><tr><td>6</td><td>7</td></tr></table>

▶その他 ― img、a

画像の表示には<img>要素を使用します。表示する画像ファイルはsrc属性で指定します。<a>要素はハイパーリンクを記述するためのものです。「ほかの文書へリンクを張ることで、関連する文書を簡単に結びつけることができる」という特徴があったからこそ、HTMLはここまで普及したといえるでしょう。

```
<a href="遷移先">リンクの文字列</a>
```

遷移先には、http://www.bing.comといったURLやファイル名を記述します。

page1.html SAMPLE page1.html

```
<!DOCTYPE html>
<html>
    <head>
        <META charset="UTF-8">
    </head>
    <body>
        <h2>ページ1：画像とリンク</h2>
        <img src="picture0.jpg" />
        <ul>
            <li><a href="http://www.google.com/">Google</a>へ</li>
            <li><a href="http://www.bing.com/">Bing</a>へ</li>
            <li><a href="page2.html">Page2</a>へ</li>
        </ul>
    </body>
</html>
```

page2.html SAMPLE page2.html

```
<!DOCTYPE html>
<html>
    <body>
        <h2>ページ2：画像とリンク</h2>
        <img src="book0.png" />
        <ul>
            <li><a href="http://www.yahoo.co.jp/">Yahoo</a>へ</li>
            <li><a href="http://www.impress.co.jp/">インプレス</a>へ</li>
            <li><a href="page1.html">Page1</a>へ</li>
        </ul>
    </body>
</html>
```

上記リストpage1.htmlのブラウザ表示結果

D:¥Book¥JavaScript¥HTML
D:¥Book¥JavaScrip...

ページ1：画像とリンク

One impress
"もっと面白い"を創造しよう。

- Googleへ
- Bingへ
- Page2へ

リンクPage2をクリック

リンク先page2.htmlに遷移

コンピュータの画像は点の集合

2-4-2 画像フォーマット

魅力的なページづくりに画像は欠かせません。画像自体は<img>要素を使って簡単に表示できますが、画像のフォーマット（形式）にはさまざまなものがあり、それぞれに特徴があることは把握しておきたいところです。

コンピュータでは画像は単なる点の集合です。点の数が多いので、そのままファイルに保存するとサイズが非常に大きくなってしまいます。最も古い形式のひとつであるBMPは、そのまま点のデータを保存するので、サイズが大きくなってしまいます。

データサイズが大きいと保存容量が必要になることはもちろんダウンロードにも時間がかかります。そこで、ファイルのサイズを削減しようとさまざまな工夫がなされてきました。現在主流となっているのはPNGとJPGでしょう。これらの特徴を次の表にまとめておきます。

PNG、JPG、BMPの特徴

形式	呼称	用途	圧縮	アルゴリズム、特徴
PNG	ピング、ピーエヌジー	アイコン、模様	可逆（元のデータに戻すことができる）	画像のラインごとに情報を圧縮。 たとえば「1100000000」のようなビット列は「11のあとに0が8回」と記録するとサイズが小さくなる。透明色が利用可能。
JPG	ジェーペグ	写真	不可逆（元のデータには戻せない）	人間が気づかない程度に情報を削除することでサイズを圧縮。特に写真画像のサイズ圧縮が得意。
BMP	ビットマップ、ビーエムピー	アイコン	なし	各ピクセル（画像を構成する小さな点）のデータをそのまま保存

既存の画像ファイルを使用する場合はあまり問題になりませんが、自分でつくった画像を保存する際には、フォーマットを選ぶ必要があります。画像に応じて適切なフォーマット選べるようになってください。

演習　実際にファイルサイズを比較してみよう

mspaint（Macでは「プレビュー」）などのアプリを起動し、お手持ちの写真を開いて（ない場合はカメラから撮影したり、ダウンロードしたりしましょう）、JPEG、PNG、BMPそれぞれの形式で保存します。JPEG、PNG、BMPのファイルサイズを比べてみましょう。また、写真の代わりに、単なる塗りつぶしの画像やアイコンなどでも同じ作業をやってみましょう。

2-4-3 | 応用例

ここまで紹介した要素だけでもいろいろなページがつくれます。

応用例1

2020年1月カレンダー

<h1>要素で見出しを表示

<img>要素で画像を表示

日	月	火	水	木	金	土
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

< table><tr><td>要素で表を表示

SAMPLE html-calendar.html

応用例2

ジャーマンポテト

材料

- ジャガイモ 2個
- 玉ねぎ 1個
- ベーコン 200g
- パセリ 少々

作り方

1. ジャガイモと玉ねぎを薄切りにします
2. ベーコン、ジャガイモ、玉ねぎの順で炒めます
3. 塩コショウで味を調えます
4. 盛り付けてパセリを振りかけて出来上がり！

で箇条書きを、<ol>で番号付きリストを表示

SAMPLE html-recipe.html

演習 HTMLで自分のページをつくってみよう

ここまで学習した要素を使って自分でページをつくってみましょう。どんなページでもかまいません。自分の手を動かすことが大切です。意図的に不正なページをつくり、ブラウザでどのように表示されるかも確認してみてください。作成したページを周囲の人に見てもらいましょう。

これだけでは、見た目が少々寂しいことは否めません。HTMLの一番の目的は文書の構造を表現することであり、ブラウザ上での表現には重きを置いていないからです。見た目はCSS（Cascading Style Sheet）で指定します。文書の構造と見た目を分離していること、これは非常に重要なポイントなのでぜひおさえておいてください。

実は、HTMLが登場した当初は、文書の構造を規定することと、文書の見映えを表現することの線引きが曖昧でした。ブラウザの開発元は、ホームページの表現力を向上させるために、新しい要素、特に見た目を充実させるための要素を競って導入していったのです。たとえば、フォントを指定する<font>要素、水平線を引く<hr/>要素、文字をスクロールさせる<marquee>要素、文字を点滅させる<blink>要素といったものまでありました。それによって、確かに見た目が派手なページを簡単に作成できるようになりましたが、その反面、ブラウザによって表示が異なってしまったり、HTML本来の目的である構造化が忘れられてしまったりと、状況は混乱の一途をたどりました。

そのような状況に危機感を抱いた標準化団体が中心となり、HTMLの表現と構造を分離する作業が進められました。その成果として、HTMLは構造に特化したシンプルなものとなり、表現はCSSで記述できるように整理されました。CSSを活用すると表現力に富んだページが作成できるようになります。本節の例で紹介したレシピやカレンダーももっと体裁よく表現できます。

2-5 統合開発環境のすすめ

みなさんはこれからたくさんのHTMLを入力することになります。効率よく入力するためのツールやアプリを持っているか否かで作業効率が大きく変わってきます。今後の作業をより効率よく進めるための統合環境について説明します。

2-5-1 統合開発環境とは

ここまでHTMLを記述するためのツールについては特に言及しませんでした。Windowsの場合はメモ帳（notepad）が一番手っ取り早いかもしれません。Windowsキーを押して「note」と入力するとメモ帳が検索されるので、それをクリックするとメモ帳が起動します。

NOTE　Macでは「テキストエディット」がお手軽です。「フォーマット」メニューから「標準テキストにする」を選択すればプレーンなテキスト形式のファイルを作成できます。

実際に筆者が高校で授業をした際も最初はメモ帳を使いました。しかしながら、メモ帳は最低限の機能しかないため、効率よくHTMLを入力することができません。タイプミスをしたために正しく表示・実行されないことも多々ありました。今後の作業をより効率のよいものにするためにも、ぜひこの機会に統合開発環境に慣れることをお勧めします。

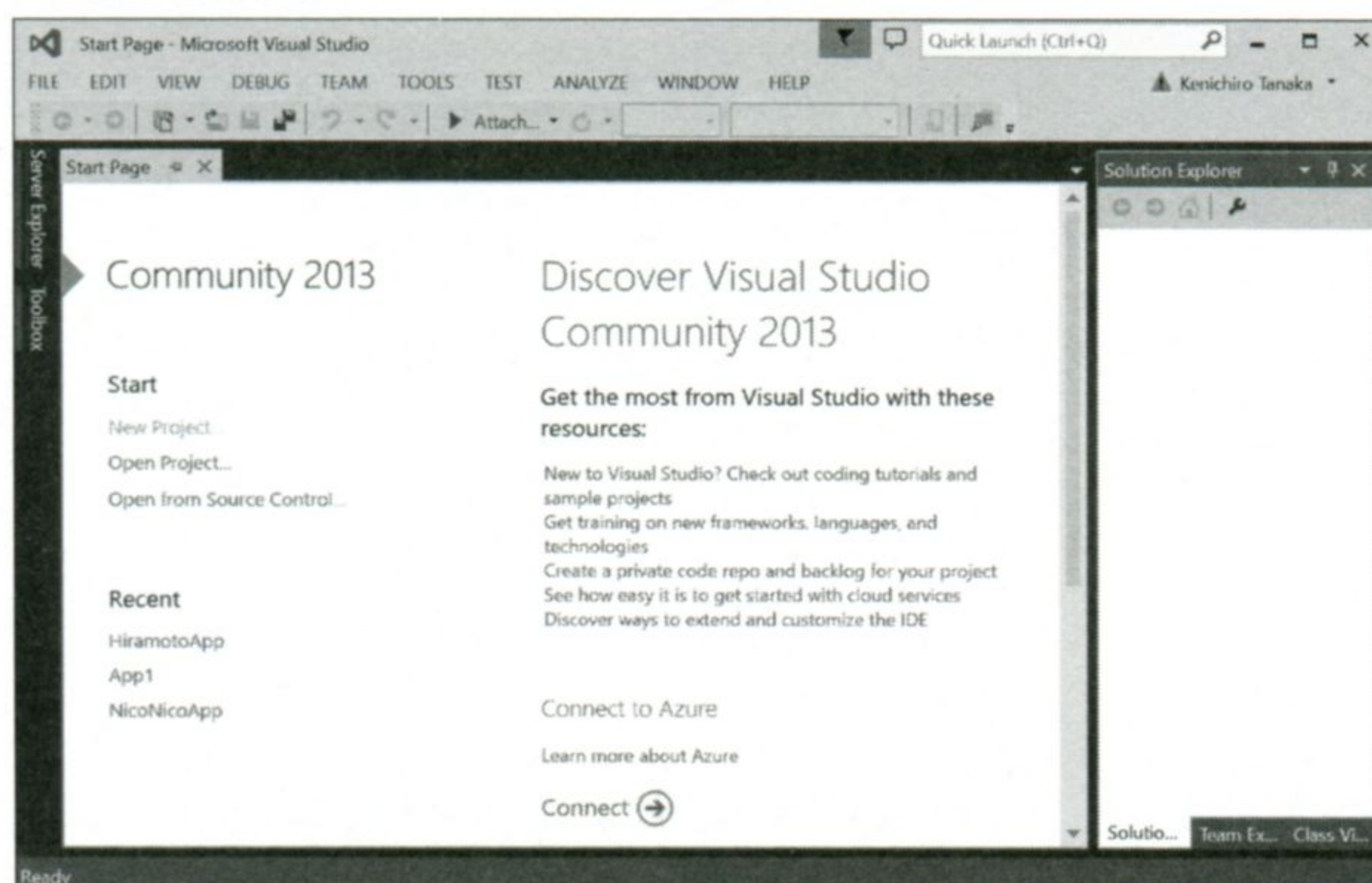
統合開発環境とはソースの編集、画面デザイン、デバッグ、テストなどアプリケーション開発に必要な機能をひとつに統合したツールです。本書の範囲においてはブラウザでデバッグすると思いますので、統合開発環境といってもHTML、JavaScript、CSSの編集を行う程度の利用に留まるかと思います。それでも開発効率は飛躍的に向上するはずです。

ここでは代表的な統合開発環境としてVisual StudioとAptana Studioを紹介します。もちろん、これ以外にもさまざまな統合開発環境があるので、評判のよいものを検索し、最新の情報を参照するようにしてください。なお、本書では開発環境のインストール手順詳細までは言及しませんが、ぜひ各自試してお気に入りの環境を見つけてください。

Visual Studio　http://www.microsoft.com/ja-jp/dev/products/community.aspx

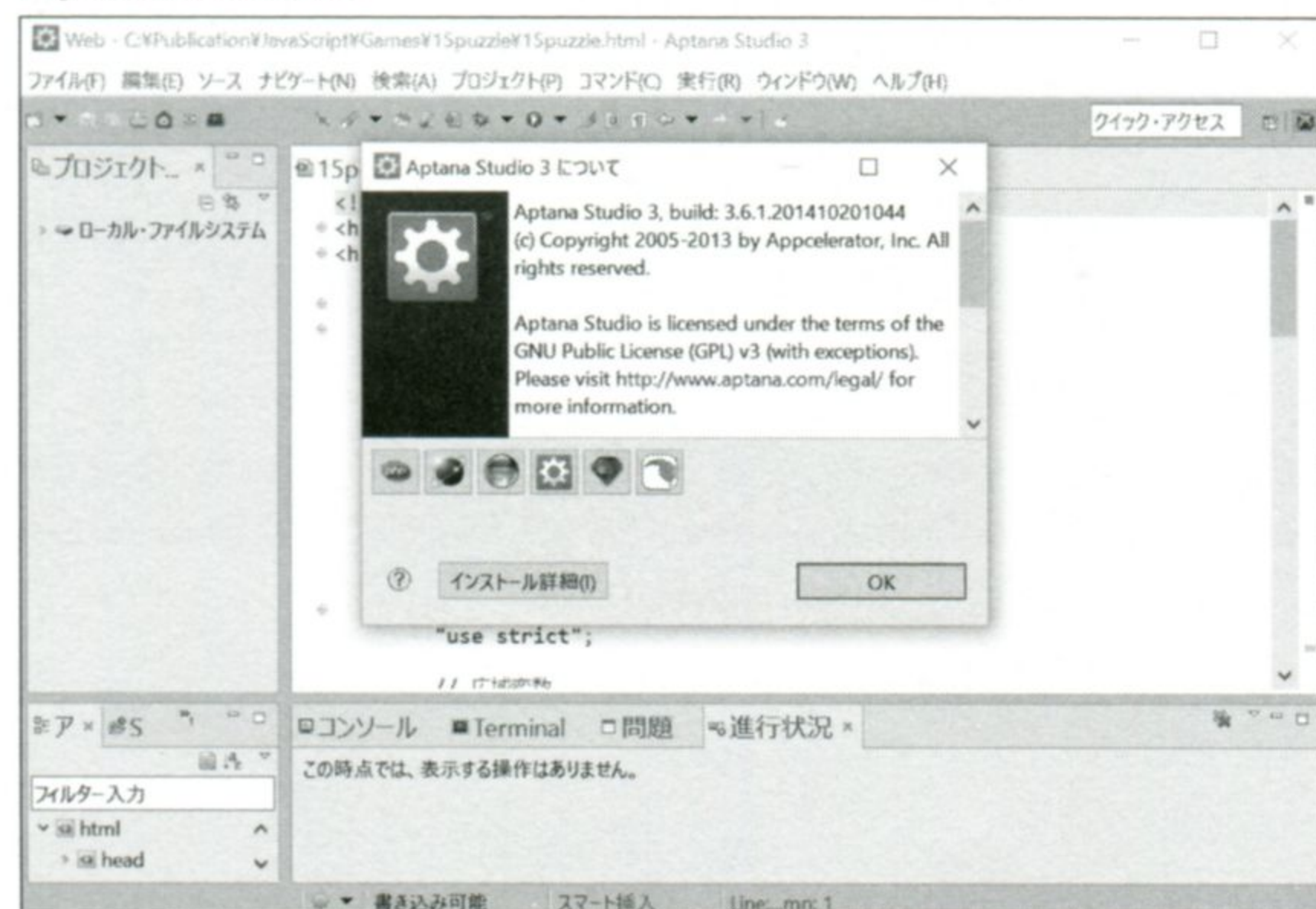
Microsoftからリリースされている統合開発環境です。用途に応じてさまざまなバージョンが用意されています。本書の範囲であれば無料版のMicrosoft Visual Studio Communityで十分です。HTMLのタグのチェック、自動補完、JavaScriptの文法チェックなどさまざまな機能を利用することができます。

Visual Studio

Visual Studio Code　https://www.visualstudio.com/ja-jp/products/code-vs.aspx

Visual Studioをベースとし、プログラムの入力に必要な機能に特化したものです。機能が限定されているため、動作も軽く、インストールも簡単です。Windows版だけでなく、Linux、OS X版も提供されています。本書の範囲では、この環境がお勧めです。

Aptana Studio　http://www.aptana.com/

単体版(Standalone)とプラグイン版（Eclipseという別の開発環境用）が提供されています。Windows版では、2015年8月時点では日本語化するためには別途プラグインをダウンロードする必要があります。

Aptana Studio

統合開発環境を使うためには、最初に操作方法を覚えなくてはなりません。特に最近の開発環境には非常に多くの機能があるため最初はとまどうかもしれません。しかし、HTML、JavaScriptでソースコードを書くだけであれば、一部の機能しか必要としないはずです。すぐに慣れることができるでしょう。ぜひ最初の壁を乗り越えて統合開発環境に親しめるようになってください。

NOTE　そのほかのMac版の開発ツールにIntelliJ IDEAなどがあります。

2-6 CSSの概要

HTMLでは文書の構造を記述します。しかし、見映えという点では物足りなさを感じられたかもしれません。HTML文書の見た目を記述するのがCSS（Cascading Style Sheet）の役割です。

2-6-1 見映えを担当するCSS

Webページでは、単に文書の内容を表示するだけでは不十分です。閲覧者が読みやすいように、文字の書体やサイズ、文字や図形の色や配置などデザインにも配慮することが大切です。このような見映えに関する設定も、昔はHTMLのタグを使って行っていました。たとえば、以下のような具合です。

SAMPLE css-basic0.html

```
<!DOCTYPE html>
<html>
    <head>
        <META charset="UTF-8">
    </head>
    <body>
        普通のテキスト
        <center>中央よせ</center>
        <font color="red">赤色のテキスト</font>
        <br />
        <font size="6">大きいテキスト</font>
    </body>
</html>
```

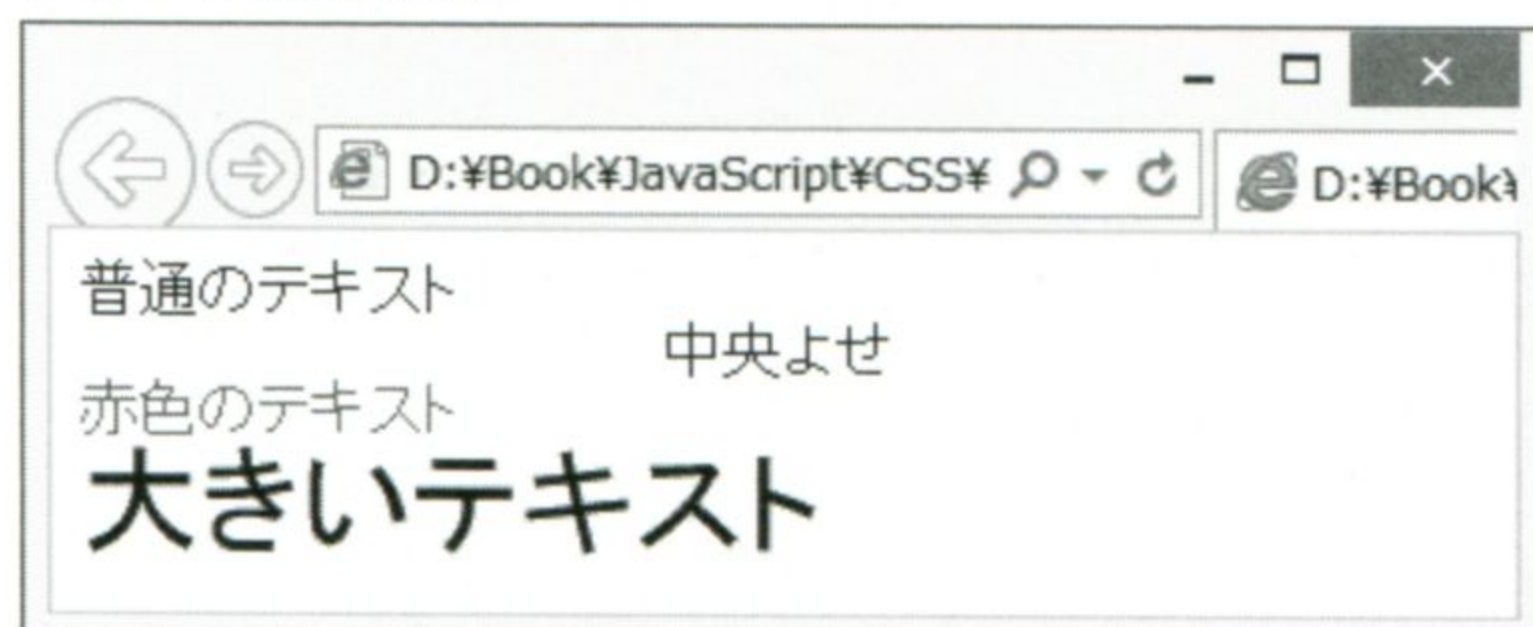
ブラウザ表示結果

D:¥Book¥JavaScript¥CSS¥

普通のテキスト

中央よせ

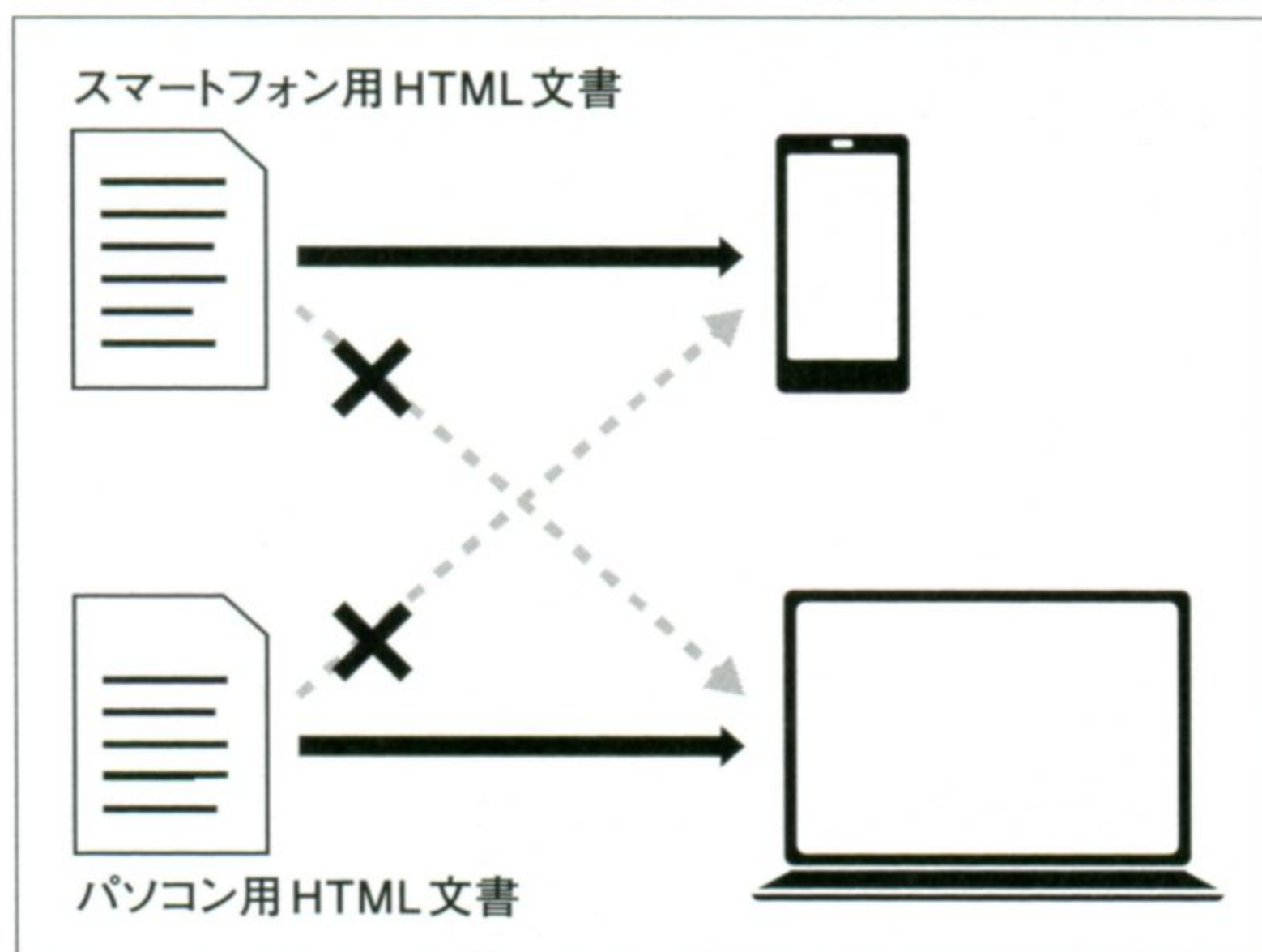
赤色のテキスト

大きいテキスト

この例のように、HTML中に見映えに関する指定を埋め込んでしまうと、その文書をいろいろな用途で使い回すときに不便です。これだけではピンとこないですよね。具体例を使って説明してみましょう。

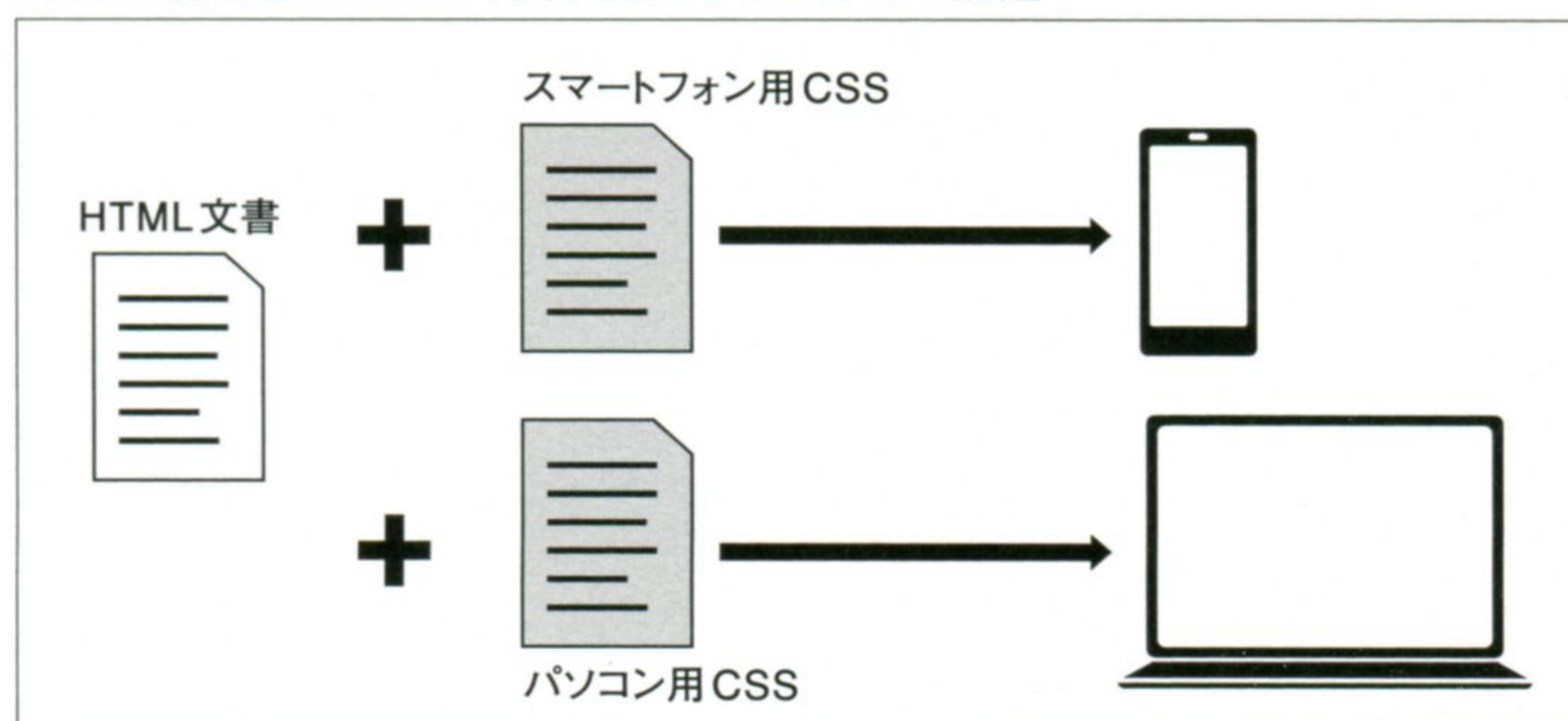
たとえば、パソコン用のページは大きな画面を想定して作成されます。レイアウトを凝ったページにして、たくさんの情報を表示することができます。一方、スマートフォンを想定している画面は、よりシンプルに、文字を

小さくする必要があります。つまり、見映えに関する指定をHTMLに埋め込んでしまうと、そのHTMLを使い回すことが難しくなってしまうのです。

HTMLで見映えを指定すると別の用途に使い回すことが難しくなる

スマートフォン用HTML文書
×
×
パソコン用HTML文書

もし、文章の構造（章、段落、見出しなど）と、見映え（色、文字の大きさ、フォントなど）を切り離すことができれば、見映えのみの修正で済むようになります。これこそが、表現と構造の分離による利点です。実際には、文章の構造をHTMLで、表現をCSS（Cascading Style Sheet）で記述します。

文書の構造をHTML、見映えをCSSに分けて記述

HTML文書
スマートフォン用CSS
パソコン用CSS

たとえば、携帯電話の画面で新聞紙面は読みづらいでしょう。同じ内容の記事でも携帯に適した文字やレイアウトになっているほうが読みやすいことは言うまでもありません。このように文書の構造と表現を切り離すと、用途別のCSSを適用するだけで、いろいろなデバイスにおいて、最適な状態で表示することが可能になります。つまり、HTML文書をほとんど修正しなくても、いろいろな用途に使えるようになるのです。

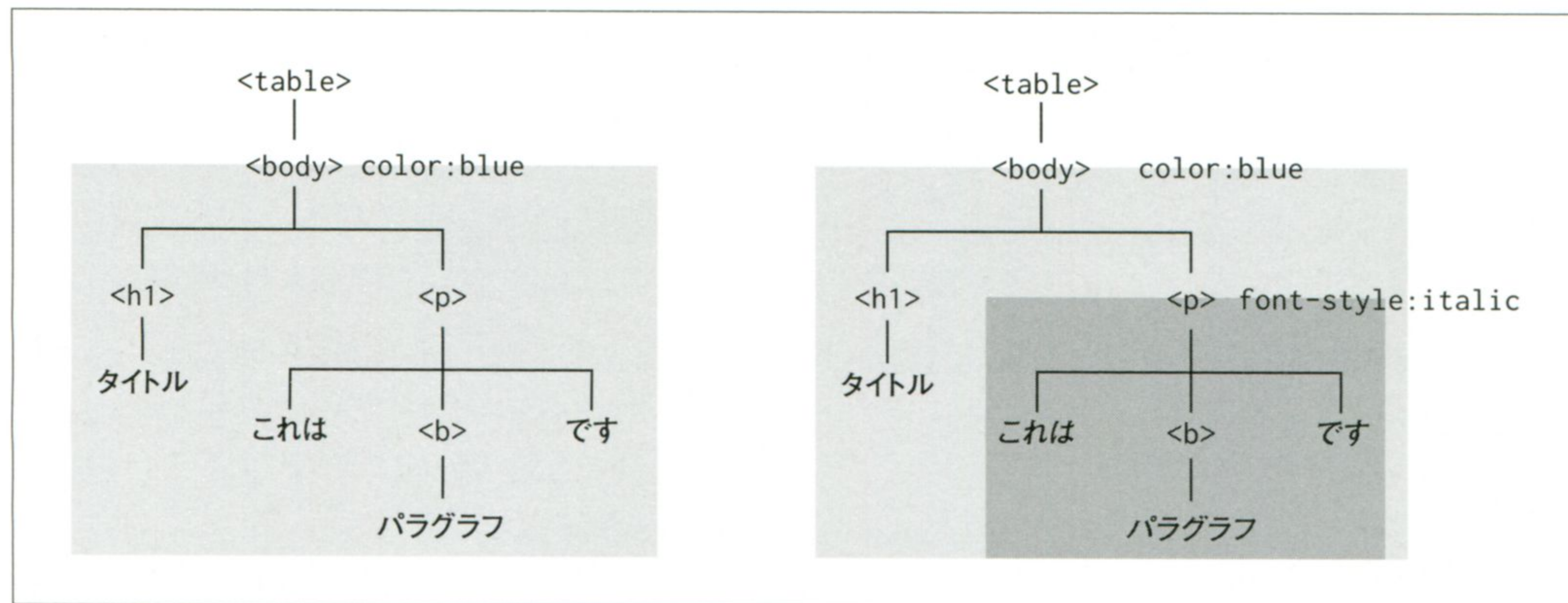
2-6-2 ｜ カスケードとは？

CSSを使うと見映えを制御できますが、文字や段落、見出しごとにフォントや色などを指定するのは面倒です。CSSには、そのような作業を軽減する便利な仕組みが容易されています。そのカギとなるのがカスケード（Cascade）です。

▶カスケードの働き

スタイルの具体例を見ていく前に、「カスケーディング」の意味を説明しておきます。カスケード（cascade）とは、滝が流れる、流れ落ちる、というような意味です。

たとえば、下図のように、body要素に「color:blue」というスタイルを適用すると、その文書全体の文字が青色になります。その子要素の<p>に「font-style:italic」というスタイルを適用すると、<p>の下にある文字がすべてイタリック（斜体）になります。

スタイルを適用した要素およびその子要素全体に適用される

つまり、スタイルは、それを適用した要素だけに有効になるのではなく、その子要素全体に適用されるのです。これが、「カスケーディング」の本質です。「上で指定したスタイルが、その下に流れていく」、そんなイメージを持っていただくとよいでしょう。このような特徴を持っているので、文書全体の見た目を一括して指定し、必要なところだけ個別にスタイルを適用することが可能となります。

ちなみに、上記のスタイルを適用した結果は、次のようになります。文書全体が青色に、段落の内容が斜体になっています。

SAMPLE css-basic1.html

```
<!DOCTYPE html>
<html>
    <head>
        <META charset="UTF-8">
    </head>
    <body style="color:blue">
        <h1>タイトル</h1>
        <p style="font-style:italic">
            これは<b>パラグラフ</b>です
        </p>
    </body>
</html>
```

ブラウザ表示結果

2-7 CSSの書き方

CSSを使うと、文字の大きさや色はもちろん、フォントの種類、行間、テキストの配置、余白、透明度などが指定できます。見た目の良いカッコいいページを作るのにCSSは必要不可欠です。ぜひ自分のものにしてください。

(2-7-1 | インラインスタイルでの指定)

それでは、このようなスタイルをどのように指定するのか、順番に見ていくことにしましょう。もっとも簡単なのは、その要素のstyle属性を使う方法です。

SAMPLE css-style0.html

```
<!DOCTYPE html>
<html>
    <head>
        <META charset="UTF-8">
    </head>
<body>
    <p style="color:red;">赤色のテキスト</p>
    <p style="color:green; font-size:24px">緑色, 24px</p>
    <p>テキストの一部にスタイルを適用する場合は、
        <span style="font-style:italic">span要素</span>を使います。
    </p>
</body>
</html>
```

ブラウザ表示結果

D:¥Book¥JavaScript¥CSS¥ | D:¥Book¥JavaScrip...

赤色のテキスト

緑色, 24px

テキストの一部にスタイルを適用する場合は、*span要素*を使います。

style属性は「style="color:green; font-size:24px;"」と例にあるように

```
CSSプロパティ名1:値1; CSSプロパティ名2:値2; …
```

と指定します。

要素によって指定できるCSS特性の値は異なります。たとえば<img>要素にfont-style特性を指定しても無意味だということは想像がつくと思います。ただ、無効な値を指定しても無視されるだけなので、最初はいろいろなCSS特性を記述して、表示がどのように変化するか試してみてください。

演習 **style属性を試してみよう**

style属性を使って文書の見た目が変わることを確認してみましょう。後述の「主なCSSプロパティ」にスタイル特性の例が書かれているので参考にしてください。

2-7-2 | CSSの主なプロパティ

以下の表に使用頻度が高いと思われるプロパティを列挙します。HTMLの要素と同じく、CSSのプロパティも膨大で、とてもすべてを覚えきれるものではありません。"たしかこんな指定ができたよなぁ…"と概要を把握しておけば十分で、詳細な使い方は必要に応じて都度調べるのがよいでしょう。

主なプロパティ

プロパティ名	用途・コメント	使用例
font-family	フォントの種類をフォント名もしくはキーワードで指定、カンマで複数の候補を指定可能	font-family: 'Times New Roman', sans-serif; font-family: Arial;
font-size	px(ピクセル)、pt(ポイント)などの絶対値や、em(フォントの高さを1とする単位)などでサイズを指定、大きさの指定方法は後述	font-size: 12px; font-size: small; font-size: large;
font-style	normal(通常)、italic(イタリック体)などフォントのスタイルを指定	font-style: italic; font-style: normal;
text-align	left(左寄せ), right(右寄せ), center(中央揃え), justify(均等割り付け)など文字の位置や割り付けを指定	text-align: center;
line-height	行の高さを指定	line-height: 22px;
color	前景色の指定、色の指定方法は後述	color: red;
background-color	背景色の指定、色の指定方法は後述	background-color: blue;
opacity	半透明の度合いを0～1の範囲で指定	opacity: 0.6;
box-shadow	矩形に影をつける効果を演出します。どのような影をつけるかを(右方向のずらし　下方向へのずらし　ぼかし具合　影の色)と4つの値で指定します。	box-shadow: 10px 10px 10px rgba(0,0,0,0.4);

CSSスタイルに関しては、Web上にも情報がたくさんありますし、書籍も充実しています。詳しくは専門のページや書籍を参照してください。

それでは、せっかく覚えたCSSプロパティを実際に使ってみましょう。ここでは、次のようなページを作ってみました。

```
<html>
    <head>
        <META charset="UTF-8">
    </head>
<body>
    <h1 style="text-align:center">安土桃山時代</h1>
    <h2 style="color:white; background-color:blue;">年表</h2>
    <table style="border: 1px solid blue; width: 600px;">
        <tr><td>1573年</td><td>室町幕府が事実上の滅亡</td></tr>
        <tr style="background-color:lightblue;"><td>1590年</td><td>豊臣秀吉日本を統一</td></tr>
        <tr><td>1592年</td><td>文禄・慶長の役</td></tr>
        <tr style="background-color:lightblue;"><td>1600年</td><td>関ヶ原の戦いで徳川家康が勝利
</td></tr>
    </table>
    <h2 style="color:white; background-color:blue;">概要</h2>
    <p>
        元亀4年（<span style="color:red">1573</span>年）に信長が
        <span style="color:blue">足利義昭</span>を京から放逐すると、
        室町幕府は事実上崩壊し、織田政権が確立する。
        信長はその後も勢力を拡大し天下統一は目前と思われた。
        しかし、<span style="color:blue">明智光秀</span>による謀反によって
        天正10年（<span style="color:red">1582</span>年）の本能寺の変で自害に至った。
</p>
    <p>
        <span style="color:blue">羽柴秀吉</span>は
        いち早く京に駆け付け首謀者である<span style="color:blue">明智光秀</span>を破った。
        天正14年（<span style="color:red">1586</span>年）には関白・太政大臣に任ぜられ豊臣姓を賜り、
        天正18年（<span style="color:red">1590</span>年）に日本を統一し
        全国で検地と刀狩りを実施させ政権の安定に力を注いだ。
    </p>
</body>
</html>
```

ブラウザ表示結果

D:¥Book¥JavaScript¥CSS¥c

D:¥Book¥JavaScript¥CS...

安土桃山時代

年表

1573年	室町幕府が事実上の滅亡
1590年	豊臣秀吉日本を統一
1592年	文禄・慶長の役
1600年	関ヶ原の戦いで徳川家康が勝利

概要

元亀4年(1573年)に信長が 足利義昭を京から放逐すると、室町幕府は事実上崩壊し、織田政権が確立する。信長はその後も勢力を拡大し天下統一は目前と思われた。しかし、明智光秀による謀反によって 天正10年(1582年)の本能寺の変で自害に至った。

羽柴秀吉は いち早く京に駆け付け首謀者である明智光秀を破った。天正14年(1586年)には関白・太政大臣 に任ぜられ豊臣姓を賜り、天正18年(1590年)に日本を統一し 全国で検地と刀狩りを実施させ政権の安定に力を注いだ。

ページの見た目は改善しましたが、HTMLのソースコードはゴチャゴチャしていますね。たとえば、人名の色と見出しの背景色を変えてほしいというリクエストがあったとしましょう。この例くらいの分量であれば、ひとつひとつのCSSプロパティを書き直してもさほどの手間はかからないかもしれません。しかし、何十ページにも及ぶ文書の場合は、かなりの作業量となるでしょう。

見た目の変更依頼に迅速に応えられないのは問題です。何が悪いのでしょうか? 見映えに関する記述が文書中に埋め込まれており、文書の構造と表現が分離できてないからです。次では、文書と見た目の分離を一段階進めてみましょう。

2-7-3 | 文書の構造と見た目の分離

CSSはHTML文書から表現を分離するために策定されたものなので、当然そのための記法が用意されています。そのカギとなるのがセレクタです。

▶Style要素での指定

HTMLでは<style>要素を使ってスタイルを一括して指定することができます。例を見てみましょう。

```
<!DOCTYPE html>
<html>
<head>
    <style>
        h1 {
            text-align: center;
        }
        h2 {
            font-style: italic;
            color: blue;
        }
        p {
            color: gray;
            font-size:14px;
        }
    </style>
</head>
<body>
    <h1>幕末における国際関係</h1>
    <h2>日米関係</h2>
    <p>
        ペリーの来航目的は補給港としての日本の開港が第一であった。
        日米和親条約に基づき、1856年8月21日（安政3年 7月21日）に
        初代米国領事タウンゼント・ハリスが来日した。
    </p>

    <p>
        このように開国初期における日本の対外関係は米国が中心であった。
        ハリスは欧州特に英国とは異なる外交路線を採用しており、
        英国公使ラザフォード・オールコックからは「幕府寄り過ぎる」とみなされることもあった。
    </p>
</body>
</html>
```

ブラウザ表示結果

D:¥Book¥JavaScript¥CSS¥ D:¥Book¥JavaScrip...

幕末における国際関係

日米関係

ペリーの来航目的は補給港としての日本の開港が第一であり、通商交渉は二義的なものとなった。結果として、1854年 3月31日（嘉永7年3月3日）に調印された日米和親条約には通商条項は含まれなかった。日米和親条約に基づき、1856年8月21日（安政3年 7月21日）に初代米国領事タウンゼント・ハリスが来日した。

このように開国初期における日本の対外関係は米国が中心であった。ハリスは欧州特に英国とは異なる外交路線を採用しており、英国公使ラザフォード・オールコックからは「幕府寄り過ぎる」とみなされることもあった。

<style>要素内でのスタイル指定は以下のように行います。

```
セレクタ {
    CSSプロパティ名: 値;
}
```

「セレクタ」とは「どの要素にスタイルを適用するか」を記述するルールです。<h1>や<p>のように要素名を書いた場合、そのタグすべてにスタイルが適用されます。適用するプロパティとその値は「プロパティ名：値；」のように記述します。プロパティ名と値の間は「：」（コロン）で区切ります。

複数のプロパティを適用したい場合は、

```
セレクタ {
    CSSプロパティ名1： 値1；
    CSSプロパティ名2： 値2；
}
```

のように、値のうしろに「;」（セミコロン）を書いて区切ります。こうすると、style要素の中を修正するだけで、文書全体の見た目を一括して更新できるようになります。たとえば、<p>のテキストをグレーから黄色に変えたい場合には「color: gray;」を「color: yellow;」に書き換えるだけで済むのです。

▶主なセレクタ

先ほどの例では要素名をセレクタとして使いました。しかし、実際に文書を作成してみると、もっと柔軟な設定をしたくなるはずです。じつはセレクタにはさまざまなルールが指定できます。ここでは代表的なセレクタの使い方をご紹介します。

全称セレクタ「*」

文書中のすべての要素に適用されます。

```
* {
    font-size: 12px;
}
```

タイプセレクタ「要素名」

指定された要素すべてに適用されます。以下の例ではすべての<h1>要素が斜体になります。

```
h1 {
    font-style: italic;
}
```

IDセレクタ「#id」

一致するid属性をもつ要素にのみ適用されます。id属性の値は文書中で一意（ほかに同じ値を持つ要素があってはならない）でなくてはならないことに注意してください。

```
#score {
    color: yellow;
}
<p>スコア：<span id="score">50</span>点</p>
```

この例では、「50」のみが黄色で描画されます。

クラスセレクタ「.クラス名」

文書中のいくつかの要素にスタイルを適用したい場合に使用します。class属性はHTMLの任意の要素に指定することができ、それらに対して一括してスタイルを適用することができます。id属性と異なり、class属性は同じ値を重複して指定してもかまいません。たとえば、以下の例では、最初と3番目のh1要素のみ青色で表示されます。

```
.bluetitle {
    color: blue;
}
<h1 class="bluetitle">HTMLの基礎</h1>
<h1>CSSの基礎</h1>
<h1 class="bluetitle">JavaScriptの基礎</h1>
```

その他のセレクタ

セレクタ	対象	例
E.クラス名	指定されたクラスを持つ要素E	p.test {color:blue;}
E:nth-child(n)	n番目の子となる要素E	p:nth-child(3) {color:blue;}
E:first-child	子として最初の要素E	p:first-child {color:blue;}
E:first-letter	要素Eの最初の文字	p:first-letter {color:blue;}

ちなみに<style>要素とセレクタを使って先ほどの例を書き直すと以下のようになります。

SAMPLE css-history2.html

```
<html>
<head>
    <META charset="UTF-8">
    <style>
        h1 { text-align: center; }
        h2 {
            color: white;
            background-color: blue;
        }
        #history {
            border: 1px solid blue;
            width:600px;
        }
        span.year {
            color: red;        ←1
        }
        span.name {
            color: blue;
        }
        tr:nth-child(2n) {        ←2
            background-color: lightblue;
        }
    </style>
</head>
```

```
<body>
    <h1>安土桃山時代</h1>
    <h2>年表</h2>
    <table id="history">
        <tr><td>1573年</td><td>室町幕府が事実上の滅亡</td></tr>
        <tr><td>1590年</td><td>豊臣秀吉日本を統一</td></tr>
        <tr><td>1592年</td><td>文禄・慶長の役</td></tr>
        <tr><td>1600年</td><td>関ヶ原の戦いで徳川家康が勝利</td></tr>
    </table>
    <h2>概要</h2>
    <p>
        元亀4年（<span class="year">1573</span>年）に信長が
        <span class="name">足利義昭</span>を京から放逐すると、
        室町幕府は事実上崩壊し、織田政権が確立する。
        信長はその後も勢力を拡大し天下統一は目前と思われた。
        しかし、<span class="name">明智光秀</span>による謀反によって
        天正10年（<span class="year">1582</span>年）の本能寺の変で自害に至った。
    </p>
    <p>
        <span class="name">羽柴秀吉</span>は
        いち早く京に駆け付け首謀者である<span class="name">明智光秀</span>を破った。
        天正14年（<span class="year">1586</span>年）には関白・太政大臣に任ぜられ豊臣姓を賜り、
        天正18年（<span class="year">1590</span>年）に日本を統一し
        全国で検地と刀狩りを実施させ政権の安定に力を注いだ。
    </p>
</body>
</html>
```
（3: `<span class="year">` の部分を指す）

2の「:nth-child(n)」はn番目の要素を指定するセレクタです。「tr:nth-child(2n)」は偶数番目の<tr>要素となります。これによって歴史年表の行が増えても偶数行にだけスタイルが自動的に付与されるようになります。

行数は以前より多少長くなりましたが、全体的にすっきりしていることがわかると思います。メンテナンスも容易になります。仮に「年号を黄色にしてくれ」と言われても、1の「color: red;」を「color: yellow;」に変えるだけで済みます。

ここで、「<span class="red">」ではなく、「<span class="year">」と、クラス名の値に「"色"」という表現ではなく、「"年"」という情報を使用していることに注目してください3。<span class="red">でもスタイルを適用することは可能です。しかしながら、文書の構造としては、年号という情報を使って文書の特定の箇所に意味を与えるべきであって、赤色という表現に関する情報を使うべきではありません。

このように<style>要素を使用することで、HTMLのコンテンツの中（<body>要素の下）からスタイル関係の記述を一切なくすことができました。文書の構造と表現の分離が大きく前進したのです。

2-8 ページのレイアウト

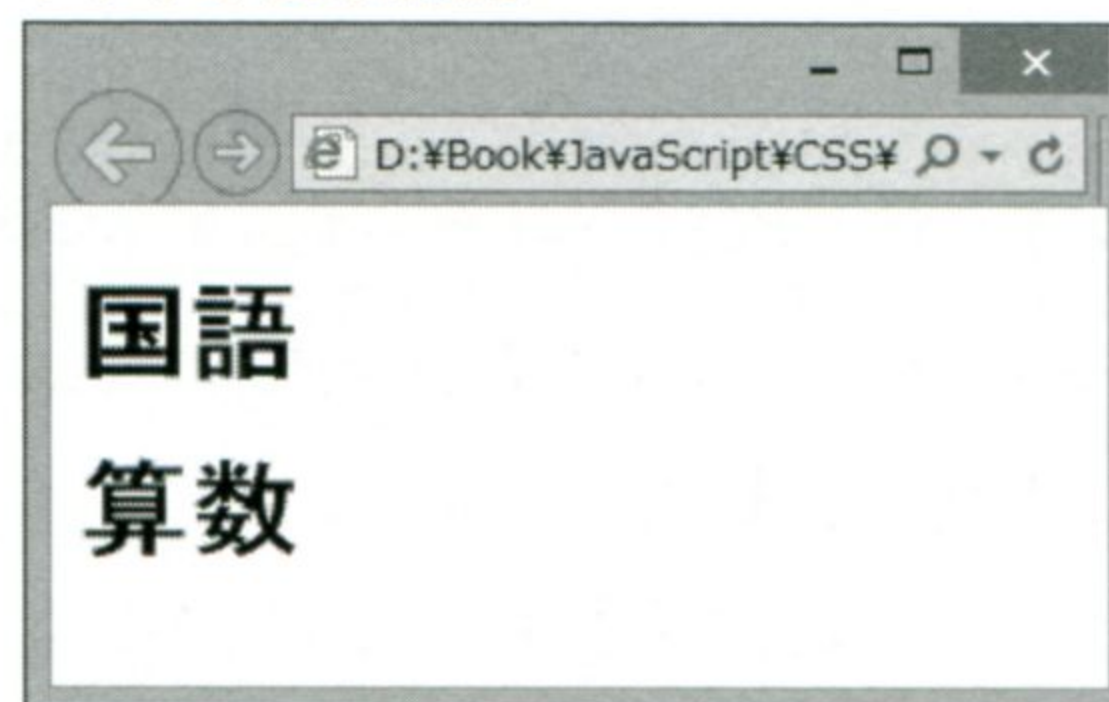
レイアウトとは配置のことです。新聞社で段組みなどの配置をきめる担当者には大きな権限が割り当てられていると聞いたことがあります。読みやすいページを作るために、配置（レイアウト）はとても大切な作業です。本節ではHTML/CSSでのレイアウトについて見ていきましょう。

2-8-1 ブロックレベル要素とインライン要素

文書の構造と表現が完全に分離され、レイアウトはCSSだけで記述できるのが理想です。しかし、実際にはレイアウトはCSSだけではなく、各要素とCSSとの共同作業で行われます。

見出し要素<h1>を連続して記述すると縦方向に並べられます。

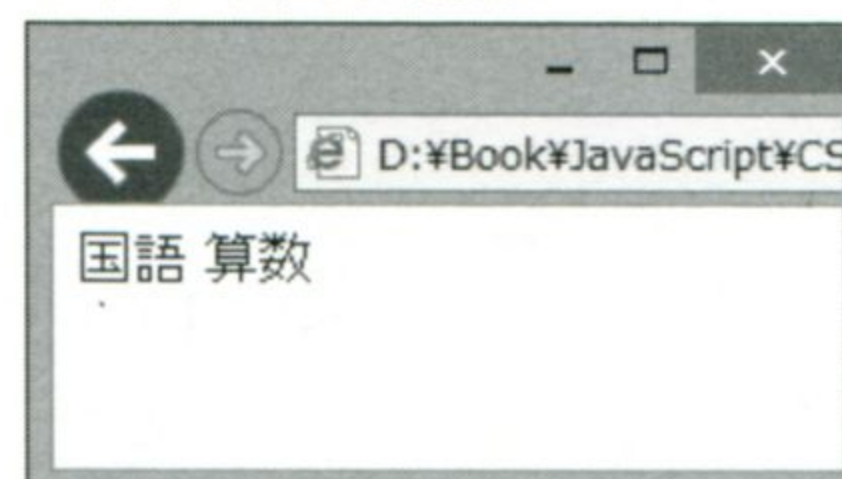
SAMPLE css-layout0.html

```
<!DOCTYPE html>
<html>
    <head>
        <META charset="UTF-8">
    </head>
    <body>
        <h1>国語</h1>
        <h1>算数</h1>
    </body>
</html>
```

ブラウザ表示結果

一方、<span>要素の場合は横方向に並べられます。

SAMPLE css-layout1.html

```
<!DOCTYPE html>
<html>
    <head>
        <META charset="UTF-8">
    </head>
    <body>
        <span>国語</span>
        <span>算数</span>
    </body>
</html>
```

ブラウザ表示結果

この違いはどこからくるのでしょうか？ 実は、HTMLの要素は大きくブロックレベル要素とインライン要素にわけられます。

▶ブロックレベル要素

<h1>のように縦方向に配置される要素です。追加されるたびに改行されます。ブロックレベル要素は高さと幅をもち、明示的に指定しない限り、幅はページ幅（もしくは親要素の幅）、高さはその中に含まれる要素にあわせられます。<h1>、<h2>、<p>、<div>、<ol>、<ul>、<li>、<table>などの要素が該当します。

縦方向にひとつづつページ幅で配置される

▶インライン要素

、<a>、<i>のように横方向に配置される要素です。文章の一部に意味を付けたり、書式を変更したりするときに使用されます。追加するだけで改行されることはありません（
は改行するための要素なので例外です）。ページ幅に入りきらなくなったときに改行が行われます。ブロックレベル要素のなかに流し込まれるようなイメージと考えるとわかりやすいでしょう。

インライン要素の中でも、<button>、<img>、<input>のように高さと幅をもつものをインラインブロック要素と呼びます。また、インライン要素がブロックレベル要素を含むことはありません。たとえば、「<p>…</p>」という記述は正しくありません。

横方向に配置されページ幅で折り返す

ページを作成してみると「スタイルを指定しているのに、思ったような場所やサイズに配置されない」という状況に何度も遭遇することになるでしょう。その多くはインライン要素とブロックレベル要素を混同していることが原因です。

たとえば、<span>要素に高さや幅を指定しても有効になりません。これは、インライン要素に高さを指定しても無視されるためです。一方、<p>要素に高さを指定するとその指定は有効になります。これは<p>要素が

ブロックレベル要素だからです。

ブロックレベル要素とインラインレベル要素によるレイアウトの様子は、要素の輪郭を表示するとよくわかります。

SAMPLE css-layout2.html

```
<!DOCTYPE html>
<html>
    <head>
        <META charset="UTF-8">
        <style>
            p, h1 {
                border: 1px dotted blue;
            }
            span {
                border: 1px solid red;
            }

        </style>
    </head>
    <body>
        <h1>HyperText Markup Language</h1>
        <p>
            HyperText Markup Language（ハイパーテキスト マークアップ ランゲージ）、
            略記・略称<span>HTML</span>（エイチティーエムエル）とは、
            ウェブ上の文書を記述するためのマークアップ言語である。
            文章の中に記述することでさまざまな機能を記述設定することができる。
        </p>
        <p>
            ウェブの基幹的役割を持つ技術の一つで<span>HTML</span>で
            マークアップされたドキュメントはほかのドキュメントへの
            <span>ハイパーリンク</span>を設定できる
            <span>ハイパーテキスト</span>であり、
            画像・リスト・表などの高度な表現力を持つ。
        </p>
    </body>
</html>
```

ブラウザ表示結果

D:¥Book¥JavaScript¥CSS¥ D:¥Book¥JavaScrip...

HyperText Markup Language

HyperText Markup Language(ハイパーテキスト マークアップ ランゲージ)、略記・略称HTML(エイチティーエムエル)とは、ウェブ上の文書を記述するためのマークアップ言語である。文章の中に記述することでさまざまな機能を記述設定することができる。

ウェブの基幹的役割を持つ技術の一つでHTMLで マークアップされたドキュメントはほかのドキュメントへの ハイパーリンクを設定できる ハイパーテキストであり、画像・リスト・表などの高度な表現力を持つ。

borderという枠を表示するCSSプロパティを指定しています。詳細は次項をご覧ください。この例をみると、ブロックレベル要素がページ幅に広がっていることや改行が行われていること、インライン要素がブロックレベル要素の中に流し込まれていることがわかると思います。

どの要素がブロックレベルで、どの要素がインラインかは使っているうちに自然に慣れてくると思いますが、まずは<p>要素、<div>要素がブロック要素であることだけを押さえてください。その際に、<p>はテキスト用、<div>はブロック要素用と把握しておけばよいでしょう。

2-8-2 ｜ ボックスモデル

前項では、ブロックレベル要素は高さと幅を持ち、デフォルトでは幅はページ幅に、高さは子要素に合わせて調整されると説明しました。では、明示的にサイズを指定したい場合はどうすればよいのでしょうか。CSSではボックスモデルとして、以下のようなCSSプロパティが用意されています。

ボックスモデル関連のプロパティ

プロパティ	説明
margin(マージン)	ほかの要素との間のスペース
border(枠線)	要素の枠線、太さ・色・スタイルなどを指定可能
padding(余白)	枠線と子要素の間のスペース
height(縦サイズ)	子要素の縦サイズ
width(横サイズ)	子要素の横サイズ

これらのCSSを使用して、幅と高さを柔軟に指定することが可能です。必要としない場合は単に省略してください。これらのスタイルを図にすると次のようになります。

ボックスモデル

marginの設定例

スタイル例	説明
margin:10px;	上下左右のマージンを一括指定
margin:10px 20px 30px 40px;	上、右、下、左のマージンを個別に指定
margin-top: 10px;	上のマージンを指定
margin-bottom: 30px;	下のマージンを指定
margin-left: 40px;	左のマージンを指定
margin-right: 20px;	右のマージンを指定

paddingの設定例

スタイル例	説明
padding:10px;	上下左右のパディングを一括指定
padding:10px 20px 30px 40px;	上、右、下、左のパディングを個別に指定
padding-top: 10px;	上のパディングを指定
padding-bottom: 30px;	下のパディングを指定
padding-left: 40px;	左のパディングを指定
padding-right: 20px;	右のパディングを指定

borderの設定例

スタイル例	説明
border-width: 2px;	ボーダー幅
border-style: solid;	ボーダースタイル、solid, dotted, dashedなど
border-color: red;	ボーダー色
border: 2px solid blue;	幅、スタイル、色をまとめて指定。border-top-style, border-left-colorなどの個別指定も可能

縦横サイズ、その他の設定例

スタイル例	説明
width:100px;	幅の指定
height:200px;	高さの指定
border-radius: 5px;	四隅のカドを丸める

2-8-3 | 色やサイズの指定

意図的にモノクロにしているページもありますが、ほとんどのページはカラーを効果的に使用しています。また、意図したレイアウトになるように、フォントのサイズや幅などを指定しています。

▶サイズの指定

これまで、フォントの大きさ、要素の位置やサイズを「100px」のようにピクセル単位で指定してきました。実は、ピクセル以外にも以下のようにさまざまな単位が利用できます。

サイズの単位

単位	説明
px	ピクセル。画面を構成する点1つ分
pt	ポイント。1pt＝1/72インチ
cm	センチメートル
vw	ビューポートの幅。画面全体の幅を100とした値
vh	ビューポートの高さ。画面全体の高さを100とした値
em	「m」のサイズ。フォントの「m」の大きさの何倍かで指定
%	パーセント。親要素のサイズの何パーセント分かで指定

▶色の指定

ここまでの例では、white、lightgreen、yellowといった名前で色を指定してきました。名前での色指定は便利な方法ですが、名前が思い浮かばなかったり、微妙なニュアンスを表現できなかったりという問題があります。

「光の三原色」は、赤（Red）、緑（Green）、青（Blue）の3つの光を組み合わせていろいろな色を表現する手法ですが、この手法は直感的にもわかりやすいのでコンピュータの世界でも広く使用されています。スタイルでも、3つの原色を使って色を指定することが可能です。赤、緑、青のそれぞれの色を0～255までの16進数で指定し、「#RRGGBB」という文字列にします。文字は大文字でも小文字でもかまいません。

16進数

10進数では0～9までの文字を使って数値を表現しますが、16進数では0～Fまでの文字を使用します。10進数と16進数の対応表を以下に示します。

10進数と16進数の対応

10進数	0	1	2	3	…	8	9	10	11	12	13	14	15	16	17	18
16進数	00	01	02	03	…	08	09	0A	0B	0C	0D	0E	0F	10	11	12

10進数	25	26	27	…	32	…	48	…	64	…	165	…	255
16進数	19	1A	1B	…	20	…	30	…	40	…	A5	…	FF

10進数から16進数の変換

10進数では、8、9、10、11…と数値が増えていきますが、16進数では8、9、A、B…と数値が増えていきます。16進数では、Fが一番大きな文字なので、そこから1つ増えると10となります。

10進数の165から16進数のA5への変換は以下のように、16で割った商と余りから簡単に求めることができます。コンピュータの世界では16進数は多用されるので、慣れていない方はこれを機に16進数に親しむようにしてください。

10進数から16進数の変換

165 ÷ 16 ＝ 10 … 5　A5

RGBを16進数に変換するとともに色を設定する例を以下に示します。まだJavaScriptの説明をしていないので、中身はわからなくても気にしないでください。

```
<!DOCTYPE html>
<html>
    <head>
        <META charset="UTF-8">
        <script>
            function setRGB() {
                var r = document.getElementById("r").value;
                var g = document.getElementById("g").value;
                var b = document.getElementById("b").value;
                var c = "#" + hex(r) + hex(g) + hex(b);
                document.body.style.backgroundColor = c;
                document.getElementById("hex").textContent = c;
            }
            function hex(v) {
                v = parseInt(v);
                var hex = v.toString(16);
                if (v < 16) { hex = "0" + hex }
                return hex;
            }
        </script>
    </head>
    <body>
        <h1 style="background-color:white">RGB:<span id="hex"></span></h1>
        <p>
        R:<input type="range" min="0" max="255" id="r" onchange="setRGB()" /><br />
        G:<input type="range" min="0" max="255" id="g" onchange="setRGB()" /><br />
        B:<input type="range" min="0" max="255" id="b" onchange="setRGB()" />
        </p>
    </body>
</html>
```

ブラウザ表示結果

Script¥CSS¥rgb-color.html D:¥Book

RGB:#ac47de

R:

G:

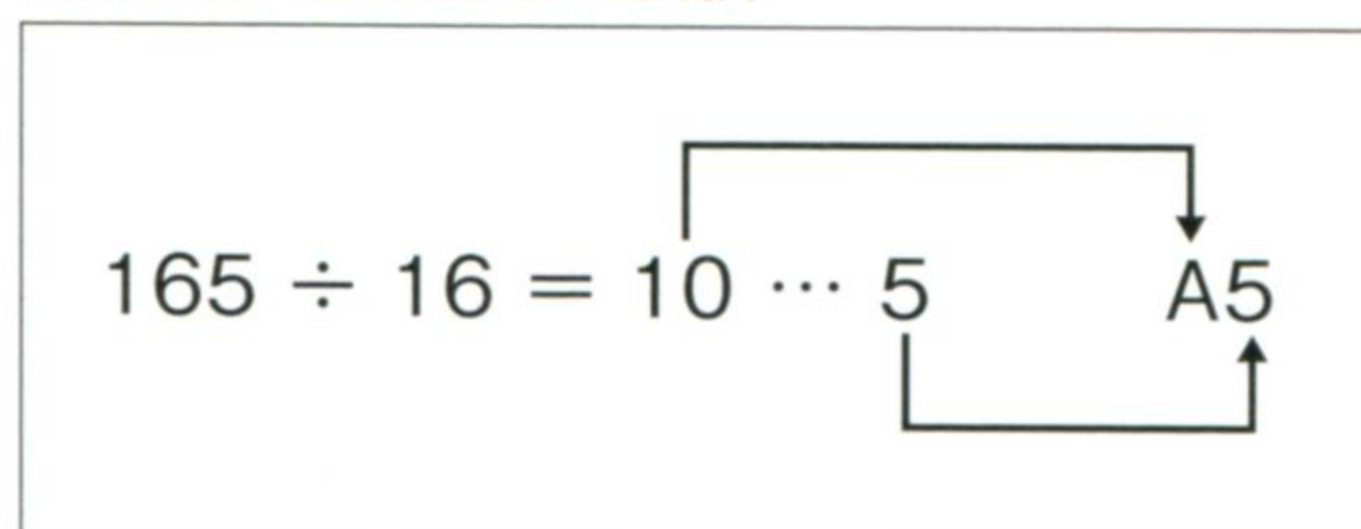
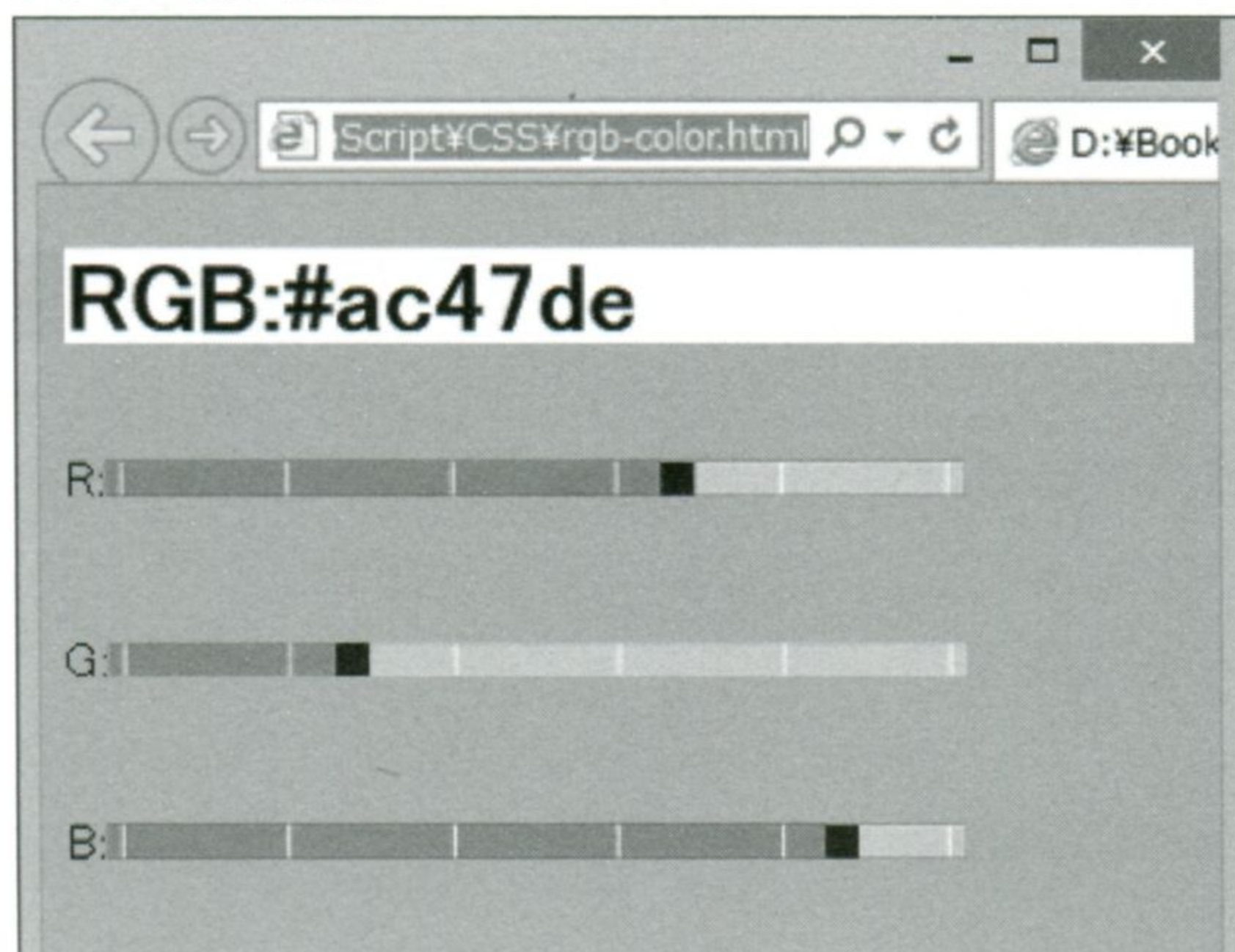
B:

ここまでのまとめとして、CSSプロパティを使って前の章で作成したカレンダーを装飾してみました。

SAMPLE css-calendar.html

```
<!DOCTYPE html>
<html>
    <head>
        <META charset="UTF-8">
        <style>
            h2 {
                color: #0094ff;
                text-align: center;
            }
            img {
                box-shadow: 10px 10px 10px rgba(0,0,0,0.4) ;
                width:500px;
            }
            td {
                font-size:28px;
                text-align: center;
                border: 1px solid #cccccc;
            }
            .red {
                color:red;
            }
            tr:nth-child(2) {
                font-weight: bold;
                font-style: italic;
            }
            td:first-child {
                color: red;
            }
            table {
                margin: 20px;
            }
        </style>
    </head>
    <body>
        <h2>2020年1月カレンダー</h2>
        <table>
            <tr>
                <td colspan="7">
                    <img src="winter.jpg" />
                </td>
            </tr>
            <tr>
```

```
        <td>日</td>
        <td>月</td>
        <td>火</td>
        <td>水</td>
        <td>木</td>
        <td>金</td>
        <td>土</td>
    </tr>
    <tr>
        <td></td>
        <td></td>
        <td></td>
        <td class="red">1</td>
        <td>2</td>
        <td>3</td>
        <td>4</td>
    </tr>
    <tr>
        <td>5</td>
        <td>6</td>
        <td>7</td>
        <td>8</td>
        <td>9</td>
        <td>10</td>
        <td>11</td>
    </tr>
    <tr>
        <td>12</td>
        <td class="red">13</td>
        <td>14</td>
        <td>15</td>
        <td>16</td>
        <td>17</td>
        <td>18</td>
    </tr>
    <tr>
        <td>19</td>
        <td>20</td>
        <td>21</td>
        <td>22</td>
        <td>23</td>
        <td>24</td>
        <td>25</td>
    </tr>
    <tr>
```

```
                    <td>26</td>
                    <td>27</td>
                    <td>28</td>
                    <td>29</td>
                    <td>30</td>
                    <td>31</td>
                    <td></td>
                </tr>
            </table>
        </body>
</html>
```

ブラウザ表示結果

D:¥Book¥JavaScript¥CSS¥c　D:¥Book¥JavaScrip...

2020年1月カレンダー

日	月	火	水	木	金	土
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

演習　**HTML、CSSを組み合わせていろいろなページをつくってみよう**

要素、セレクタ、スタイル、この章で学習したことを活用して、時間割、ToDoリスト、学級新聞など、好きなページをつくってみてください。本書で紹介したCSSスタイルはごく一部です。ほかの書籍やネットの情報を参考にしながら、いろいろなスタイルを試してみましょう。

JavaScriptの基本

Chapter 3

HTML5
CSS
JavaScript
Canvas
Game
and
Physics engine

JavaScriptは数あるプログラミング言語のひとつです。ブラウザさえあれば試してみることができます。かつては敷居の低さから入門用言語とみなされていましたが、最近ではその良さが改めて見直されつつあります。Webサイト制作では一番利用されている言語でもあり、動的なページをつくるにはある程度の知識は必須といえるでしょう。

3-1 プログラミング言語JavaScript

JavaScriptは1995年にブレンダン・アイク（Brendan Eich）によって開発されました。名前が似ているためにJavaと混同されることもありますがまったく別の言語です。さぁ、JavaScriptの冒険の旅にでかけましょう!

3-1-1 | プログラミング言語とは

そもそもプログラミング言語とは何でしょうか？ いろいろな定義があると思いますが、ここでは「コンピュータに命令するための言葉」と考えましょう。英語を話せると世界中の多くの人とコミュニケーションできるように、プログラミング言語をマスターするとコンピュータに命令できるようになります。

たとえば、一般的なリアルタイムゲームの処理手順は以下のようになります。

① いろいろな用意（初期化）をする
② キャラクターの移動
　ア）敵の移動
　イ）自分の移動（マウス、タッチ、キーボードの入力をチェック）
　ウ）衝突判定（衝突時はゲームオーバー）
③ 画面の更新
　ア）画面をクリア
　イ）背景、キャラクター、敵などを画面に描画する
④ 手順②に戻る

上の処理手順は日本語で書かれていますが、これをコンピュータにわかるようにプログラミング言語で表現すると、実際にゲームがコンピュータ上で実行されます。

世の中にはたくさんのプログラミング言語が存在しますが、動的ページを実現するための言語としてひろく使用されていること、どのOSでも手軽に試すことができることなどから、本書ではJavaScriptという言語を採用することにしました。

プログラミング言語を駆使すると複雑な処理ができますが、ひとつひとつの命令は非常にシンプルです。主な処理命令は以下のとおりです。

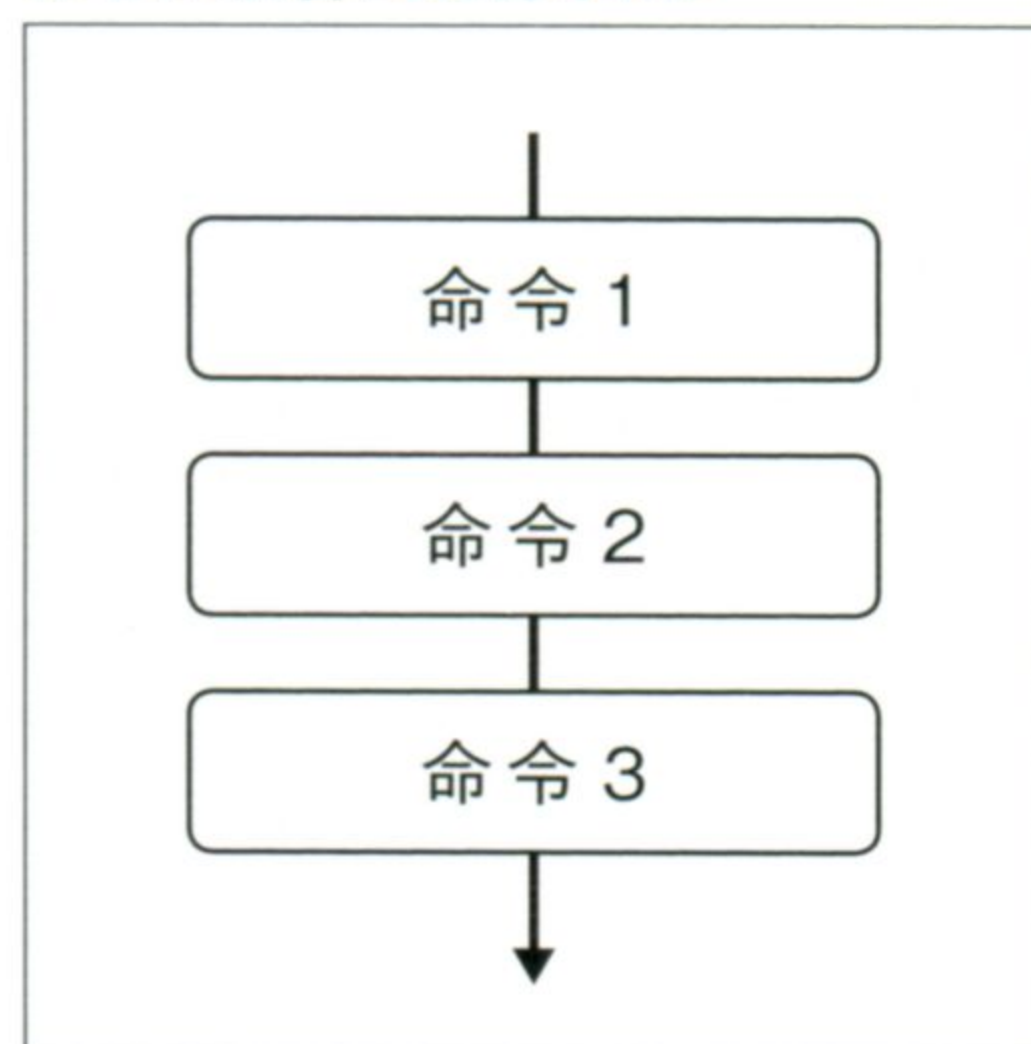
- 代入：「変数」という箱に数値や文字を格納する
- 演算（計算）：四則演算（足算、引算、掛算、割算）を行う
- 比較：ふたつの値が同じかどうか、大小関係を比べる
- 条件分岐：比較した結果で処理の手順を変える
- 繰り返し：同じ処理を繰り返す

次節から、これらの基本的な命令をJavaScriptではどのように記述するか見ていきますが、その前にまず、全体的な処理の流れを把握しておきましょう。

3-1-2 | JavaScriptのプログラム実行の流れ

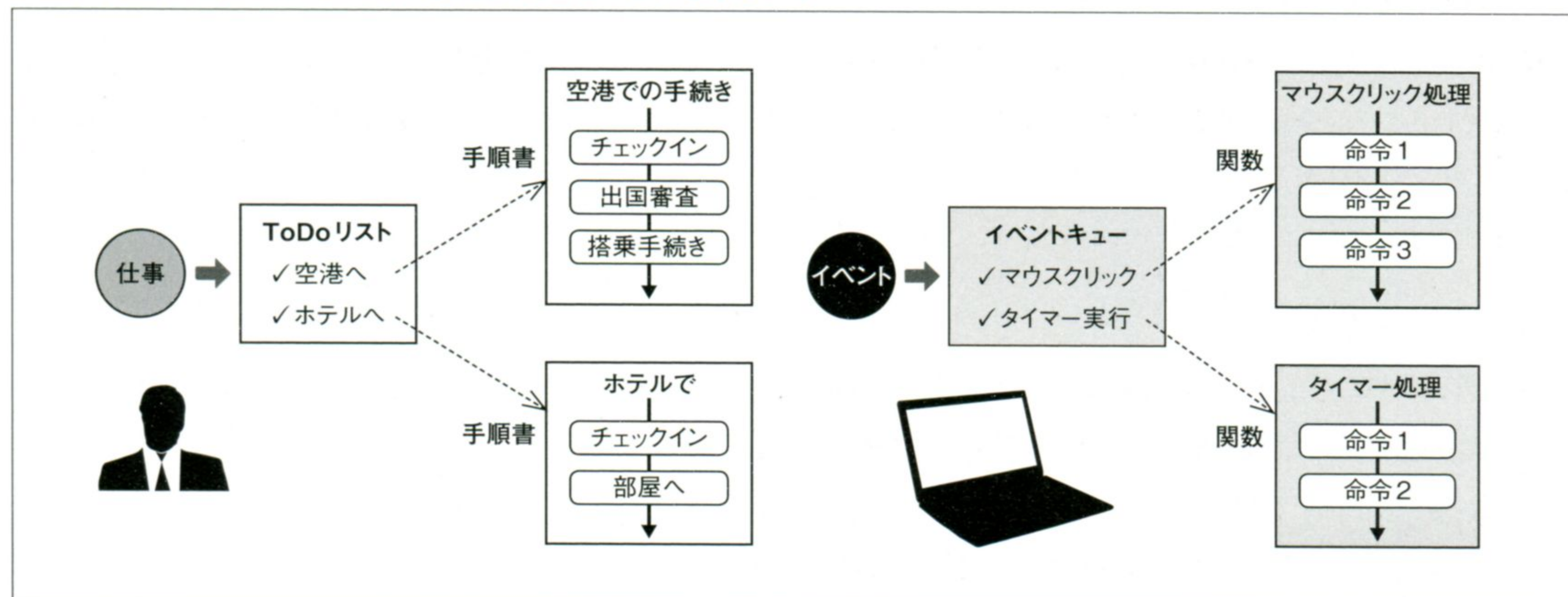
基本的にプログラムは記述された順に、すなわち、上の行から下の行へと順番に実行されていきます。

命令は順番に実行される

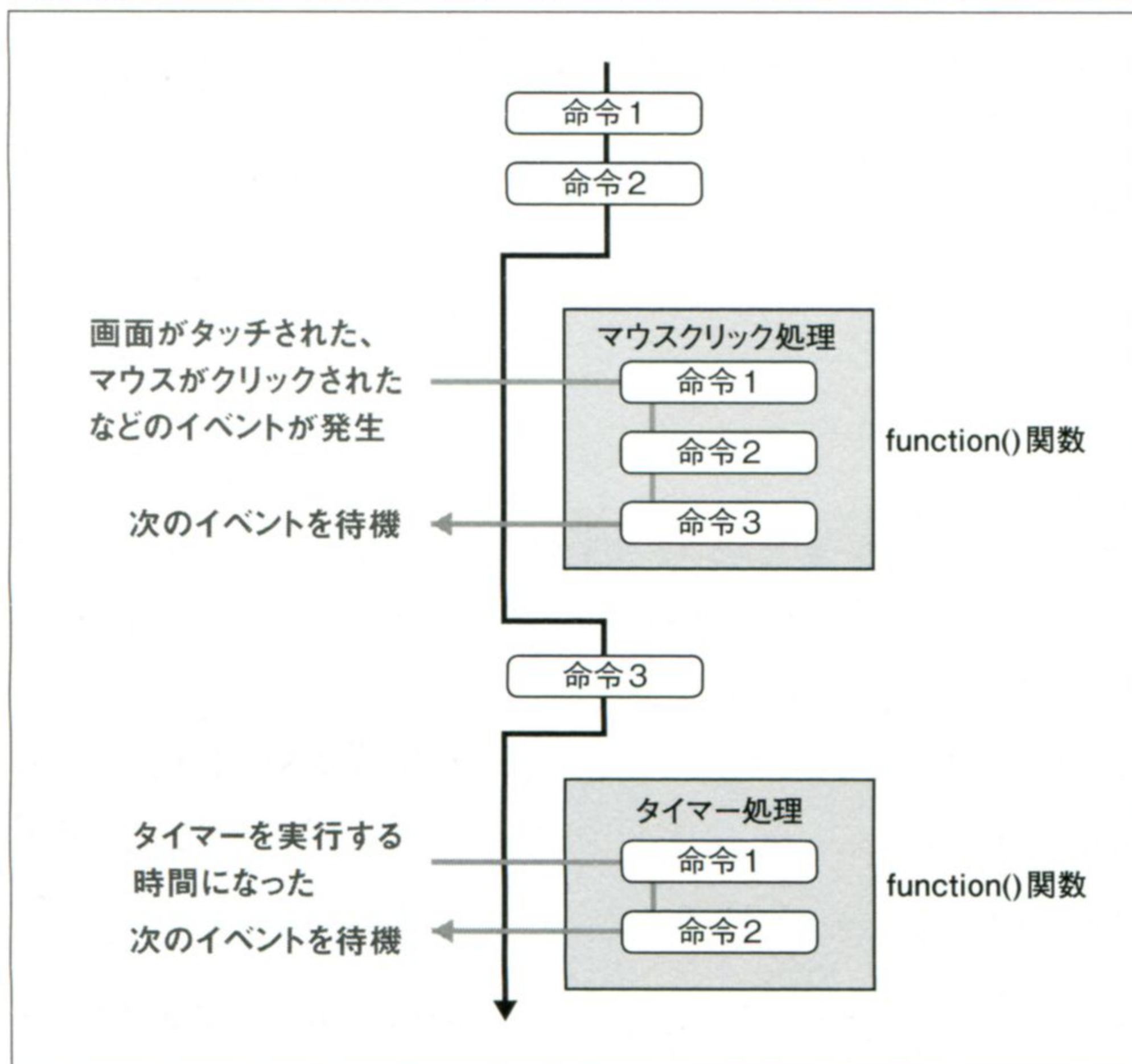
命令1
命令2
命令3

さらに、JavaScriptの場合、ユーザーからのマウスクリックを処理したり、一定時間ごとに実行されるタイマー処理を実行したりなど、何らかの出来事（イベント）が起きたときの処理を事前に記述しておきます。

例を使って説明しましょう。ToDoリストですべての仕事を管理している人がいたとします。出張や会議など業務が発生したら、その業務はToDoリストに追加されます。その人はToDoリストに追加された順に、ひとつひとつ業務をこなしていきます。それぞれの業務には手順書が用意されており、その人は忠実にその手順を実行します。ある業務が終わったら、ToDoリストにある次の仕事に取りかかります。ToDoリストが空になったらやっと休憩です。

プログラムの流れはToDoリストを順番に処理するのと似ている

仕事
ToDoリスト
✓空港へ
✓ホテルへ
手順書
空港での手続き
チェックイン
出国審査
搭乗手続き
手順書
ホテルで
チェックイン
部屋へ
イベント
イベントキュー
✓マウスクリック
✓タイマー実行
関数
マウスクリック処理
命令1
命令2
命令3
関数
タイマー処理
命令1
命令2

JavaScriptのプログラムもこれと同じように処理されていきます。ユーザーからの入力、タイマー実行などの出来事（イベント）は「イベントキュー」というToDoリストに追加されます。ブラウザは、イベントキューに追加された順にひとつひとつの仕事をこなしていきます。それぞれのイベントには「関数」と呼ばれる手順書が用意されており、ブラウザはその関数を実行します。関数の実行が終わったら、次のイベントの処理に取りかかります。もし、手順書が見つからなかった場合、そのイベントは単に無視されます。イベントキューが空になったらやっと休憩です。

JavaScriptでは、関数は「function」というキーワードで記述します。つまり、これがJavaScriptでの手順書となります。functionの外側の部分は、最初にページが読み込まれたときに1回だけ上から順番に実行されます。functionの内側の部分はイベントが処理されるとき、先の例でいえばToDoリストの仕事を実行するときに実行されます。そのイメージを以下の図に示します。

functionの外側は最初に1回だけ、functionの内側はイベントに応じて実行される

ちなみに、イベントに対応付けられた関数は「イベントハンドラ」と呼ばれます。

いきなり、イベント、関数、functionなどの用語がでてきたので、とまどった人もいるかもしれません。でも、のちほど詳しい説明があるので心配はご無用です。ここでは、

- JavaScriptのプログラムではイベントに対応する関数functionを記述しておく
- 新しい出来事（イベント）はToDoリストに追加される
- ブラウザはToDoリストに追加された仕事を順番に取り出し、イベントに対応する手順書functionがある場合にはそれを実行する

ということを理解してください。

3-2 変数と演算

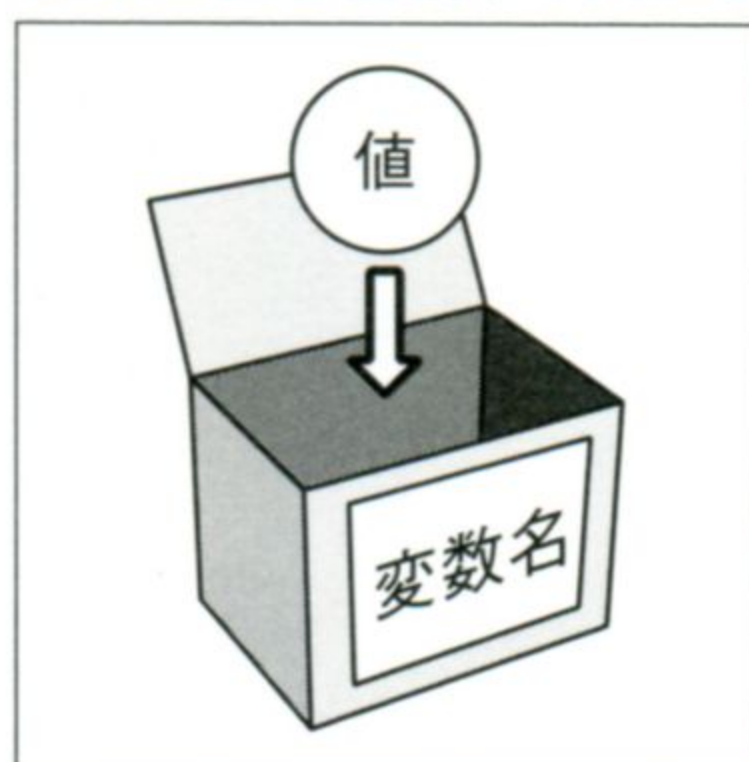
コンピュータは「計算機」と訳されることもあるくらい、計算は最も得意とするところです。プログラミングでも計算は欠かすことはできません。変数に値を格納し、その変数に対して演算子を使うことで加減乗除といった計算を行います。直観的でわかりやすい処理だと思います。

3-2-1 変数の宣言

変数とは値を格納する入れものです。入れものには名前を付けられますが、それを「変数名」と言います。中学校の数学でxやyを使って方程式を解いたと思いますが、このxやyこそが変数にほかなりません。

変数は値を格納する入れもの

値

変数名

JavaScriptでは変数を使用する前に、どんな変数を使用するのか事前に宣言する必要があります。そのためには、varというキーワードのあとに空白を挿入し、次に変数の名前を書きます。複数の変数を指定する場合は、変数名をカンマで区切ります。これを「変数の宣言」と言います。

NOTE 本当は宣言しなくても使用できるのですが、バグの原因ともなるので必ず宣言する癖をつけるようにしてください。

```
var s;                  // 変数sを宣言
var r, score, point;    // カンマ区切りで複数の変数を宣言
```

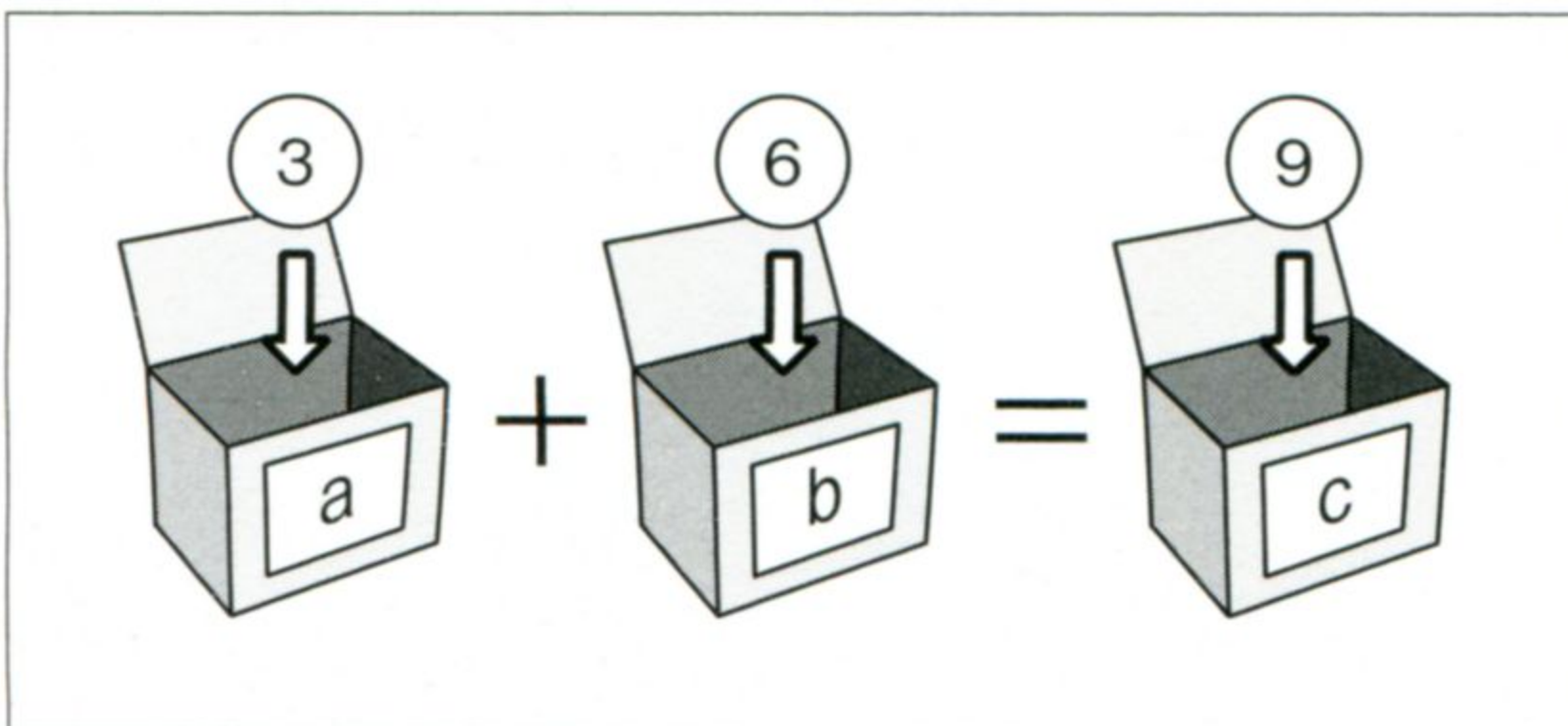
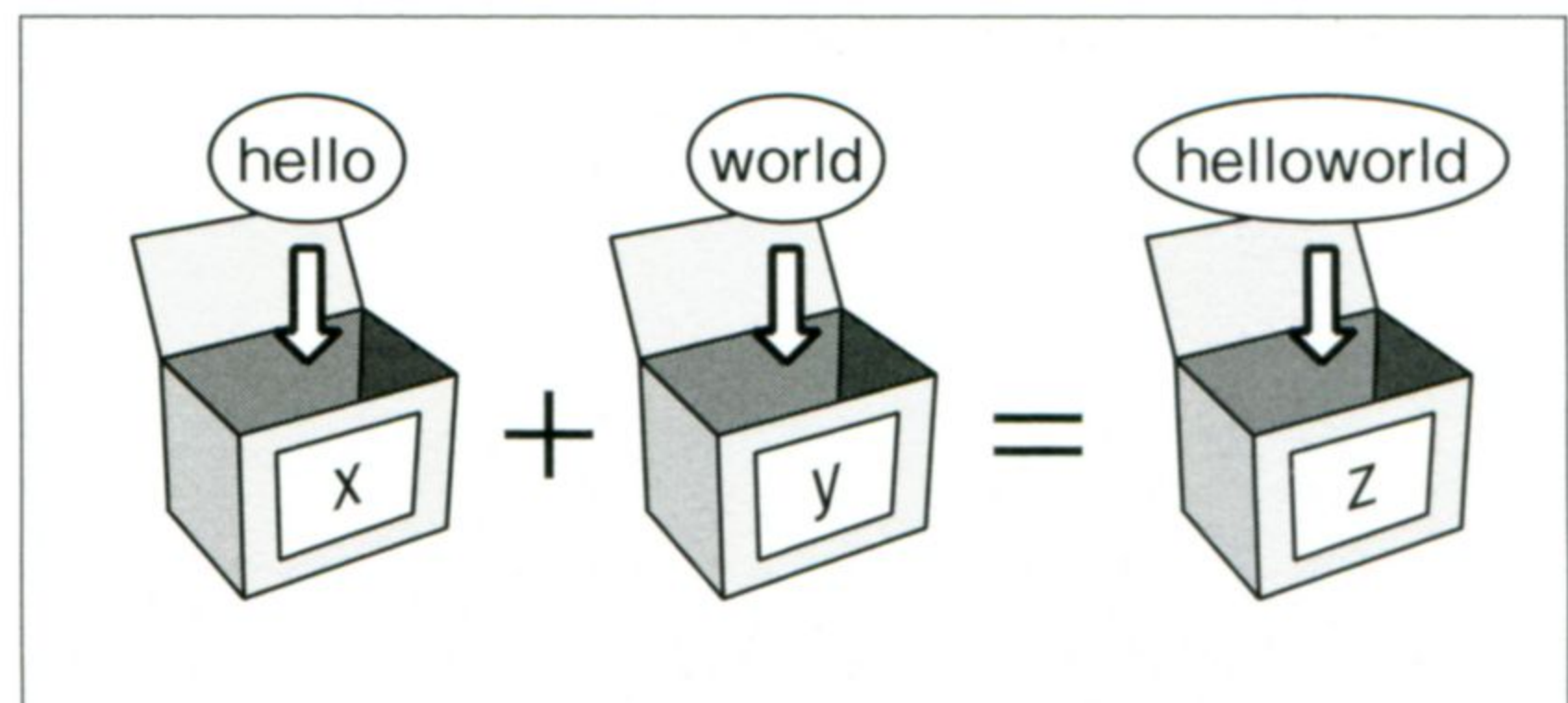
変数を宣言するときに、「= 値」と書くことで、最初の値（初期値）を設定することができます。あとで見直すときのことを考えて、変数名にはわかりやすい名前をつけるよう心掛けてください。

JavaScriptの変数には、数値、文字などいろいろなものを格納することができます。数値はそのまま数を記述しますが、文字列（文字の並び）は" "もしくは' 'で囲みます。

SAMPLE calculation.html

```
var a = 3;        // 変数aを宣言し、「3」を格納       ←「//」以降はコメント
var b = 6;        // 変数bを宣言し、「6」を格納
var c = a + b;  // 加算   c = 9
var d = a - b;  // 減算   d = -3
var e = a * b;  // 乗算   e = 18              ←1
var f = a / b;  // 除算   f = 0.5

var x = "hello";
var y = "world";
var z = x + y;  // z = "helloworld" ←2
```

変数に数値が入っている場合は1のように四則演算が可能です。文字列の場合、足し算をすると2のように文字列が連結されます。

数値の足し算

文字列の足し算

▶コメント

コメントは文字どおり「注釈」という意味です。プログラムの実行には影響を与えません。ほかの人への説明だったり、自分が忘れないためのメモ書きだったり、いろいろな用途に利用できます。JavaScriptでのコメントの書き方は以下の2とおりがあります。

1行コメント「//」

スラッシュ「/」を2個連続するとその後ろから行末までがコメントになります。

```
var a = 3;  // このうしろから行末までがコメントになります
```

複数行コメント「/* ～ */」

「/*」(スラッシュとアスタリスク)から「*/」までの部分がコメントとなります。

```
/*
  ここに複数行にわたるコメントを書きます。
  ここに複数行にわたるコメントを書きます。
  ここに複数行にわたるコメントを書きます。
*/
```

コメントは非常に便利なツールです。しかし、コメントを濫用しないよう注意してください。シンプルで誰が見てもよくわかる、コメントがなくてもその意図がはっきり伝わる、そんなコードが理想です。当たり前のことをコメントで記述したり、コメントがないと理解不能だったり、というような状況に陥らないように心がけましょう。

3-2-2 | 演算

加算は「+」記号、減算は「−」記号です。これらは直感的にわかると思います。プログラムでは、掛け算の×は「*」（アスタリスク）、割り算の÷は「/」（スラッシュ）という記号を使います。「=」は値を代入するときに使います。

```
var a = 6, b = 2;
var c = a * b;  // c = 12
var d = a / b;  // d = 3
```

余りを求める場合は「%」を使用します。

```
var a = 5, b = 2;
var c = a % b;  // c = 1  5÷2の余りは1
```

また、プログラムでは値を1増やしたり、逆に1減らしたりすることをよく行います。そこで、専用の書き方が用意されています。

```
var a = 5, b = 3;
a++;      // aの値を1増やす→aは6になる
b--;      // bの値を1減らす→bは2になる
```

ほとんどの場合、計算結果は変数に代入することになります。

```
var a = 3;
a = a + 2;  // この行を実行したあと、aは5となる
```

読者のなかには「a = a + 2;」という記述に違和感を覚えるかもしれません。方程式として考えると、この式はなりたちません。プログラミング言語の場合は、イコールの右辺を計算して、その結果を左辺に代入する、という処理手順になります。

この例の場合、aは3で初期化されています。よって、右辺のa + 2は5となり、それが再びaに代入されます。つまり、この行の実行が終わるとaの値は5になります。

変数自身の値を使って計算し、その結果を自身に代入するという処理は頻繁に行われるので、以下のように簡便な記法が用意されています。

```
var a = 3, b = 3, c = 3, d = 3;
a += 2;     // a = a + 2; aは5になる
b -= 1;     // b = b - 1; bは2になる
c *= 4;     // c = c * 4; cは12になる
d /= 2;     // d = d / 2; dは1.5になる
```

式と値

式というと「2+3=5」といった計算式が思い浮かぶでしょう。しかし、プログラミング言語では、計算式だけではなく、実行した結果、最終的に値が求まるもの全般を意味し、たとえば、単なる値や変数だったり、一定の処理を実行したりすることまで式に含まれます。

3-3 比較と条件式

比較とは文字どおりふたつの値を比べることです。条件式とはその比較結果に応じて処理の流れを変えるための命令です。比較と条件式はプログラミングにおいて最も基本的で大切な内容のひとつです。ぜひしっかりと習得してください。

3-3-1 | 比較した結果に応じて処理を変える

ゲームを実行していると毎回いろいろなことが起こります。たとえば、神経衰弱ゲームでは、同じ数字のカードを開いたらそのカードは開いたままになりますが、違う数字だったらカードは再び裏返しです。シューティングゲームで自分の弾が敵にあたったら敵を撃破します。外れたときは何も起こりません。落ちモノ系ゲームで同じ色のブロックが3個以上つながったらそれらを消去しますが、3個未満のときは何もしません。

ゲームの処理判断の例

ゲーム	神経衰弱	シューティング	ブロック
比較するもの（条件）	ふたつのカードの数字が同じかどうか?	自分の弾が敵に当たったかどうか?	3つ以上同じ色が並んだか?
条件成立時の処理	開いたまま	敵を撃破	ブロックを消去
条件不成立時の処理	元の裏返し	弾の位置をすすめる	何もしない

このように、ゲームでは時々刻々変化する状況に応じて処理を変えていく必要があります。
これらの例に共通しているのは、

- 何らかの比較をして
- その結果に応じて処理を変える

ということです。「比較して、その結果に応じて処理を変える」、これはプログラミングにおいて最も基本的な処理のひとつです。ゲームにおいても同じです。いつも決まったことが起きるようではゲームにならないですよね。

▶条件式

では、比較処理から見ていきましょう。比較処理を行うための命令を「条件式」と呼びます。条件式を実行するとtrue（条件が成立した場合）かfalse（条件が不成立の場合）が結果として返されます。主な条件式は以下のとおりです。

条件式

条件式	説明	a = 3, b = 3 の場合	a = 3, b = 5 の場合
a == b	aとbの値が同じ	true	false
a != b	aとbの値が違う	false	true

条件式（前ページの続き）

条件式	説明	a = 3, b = 3 の場合	a = 3, b = 5 の場合
a < b	aよりbのほうが大きい	false	true
a > b	aのほうがbより大きい	false	false
a <= b	aはb以下	true	true
a >= b	aはb以上	true	false

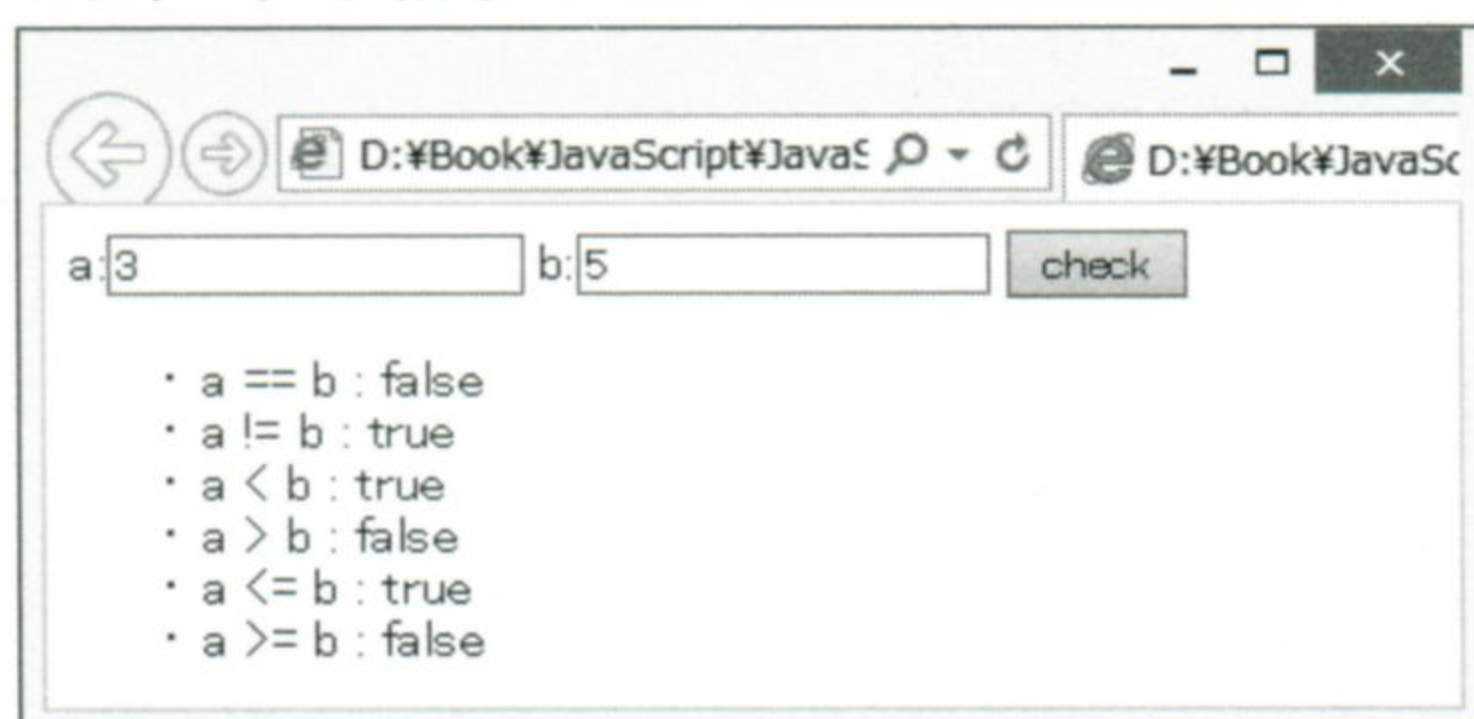
条件式の挙動を実際に確認するページを作ってみました。

条件式を利用した例 SAMPLE condition.html

```
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            function check() {
                var a = document.getElementById("i0").value;                    ←1
                var b = document.getElementById("i1").value;
                document.getElementById("r0").textContent = (a == b);   ←2
                document.getElementById("r1").textContent = (a != b);
                document.getElementById("r2").textContent = (a < b);
                document.getElementById("r3").textContent = (a > b);
                document.getElementById("r4").textContent = (a <= b);
                document.getElementById("r5").textContent = (a >= b);
            }
        </script>
    </head>
    <body>
        a:<input id="i0" value="3" />
        b:<input id="i1" value="5" />
        <button onclick="check()">check</button>
        <ul>
            <li>a == b : <span id="r0"></span></li>
            <li>a != b : <span id="r1"></span></li>
            <li>a < b : <span id="r2"></span></li>
            <li>a > b : <span id="r3"></span></li>
            <li>a <= b : <span id="r4"></span></li>
            <li>a >= b : <span id="r5"></span></li>
        </ul>
    </body>
</html>
```

ブラウザ表示結果

a: 3 b: 5 check

・a == b : false
・a != b : true
・a < b : true
・a > b : false
・a <= b : true
・a >= b : false

HTMLにおいて、<input>要素に入力された値を取得するには❶のように

```
var 変数 = document.getElementById("<input>要素のID").value;
```

という命令を使います。結果を画面に表示するには❷のように

```
document.getElementById("表示する要素のID").textContent = 表示内容;
```

という命令を実行します。<button>要素がクリックされると、その要素のonclick属性で指定された内容、前ページの例では関数check()が実行されます。input要素から値を取得したり、画面に値を表示したりする方法については、「3-7-3 JavaScriptからHTMLを操作する」(P.109)で説明しますので、ここでは「そのようなものか」と思ってください。今は、条件式について慣れることを優先してください。

演習　前記のHTMLを入力して実行してみよう

いろいろな数値を入れて条件式の評価結果がどうなるか実際に試してみましょう。

(3-3-2 | 条件式 ― if文)

条件式を評価するとtrueかfalseが返されることを見てきました。次に、その結果に応じて処理を切りかえる「制御式」について見ていきましょう。最もよく使われるのがif文です。if文は以下のように記述します。

if文の書式

```
if (条件式) {
    命令文1;
} else {
    命令文2;
}
```

条件式がtrueのときに命令文1を実行し、falseのときに命令文2を実行します。命令文2の実行が不要なときはelse以降を省略してもかまいません。以下の例では、条件式がtrueのときに命令文1が実行されます。

```
if (条件式) {
    命令文1;
}
```

逆に、「Aの場合は命令1、Bの場合は命令2、Cの場合は命令3」…といったように条件が複数ある場合には、else ifを使って条件式と命令文のペアを必要なだけ連続して記述することが可能です。

```
if (条件式1) {
    命令文1;
} else if (条件式2) {
    命令文2;
} else {
    命令文3;
}
```

この例では、条件式1がtrueのときは命令文1が、条件式2がtrueの時は命令文2が、それ以外のときは命令文3が実行されます。簡単な例を見てみましょう。

if文を利用した例 SAMPLE ifelse.html

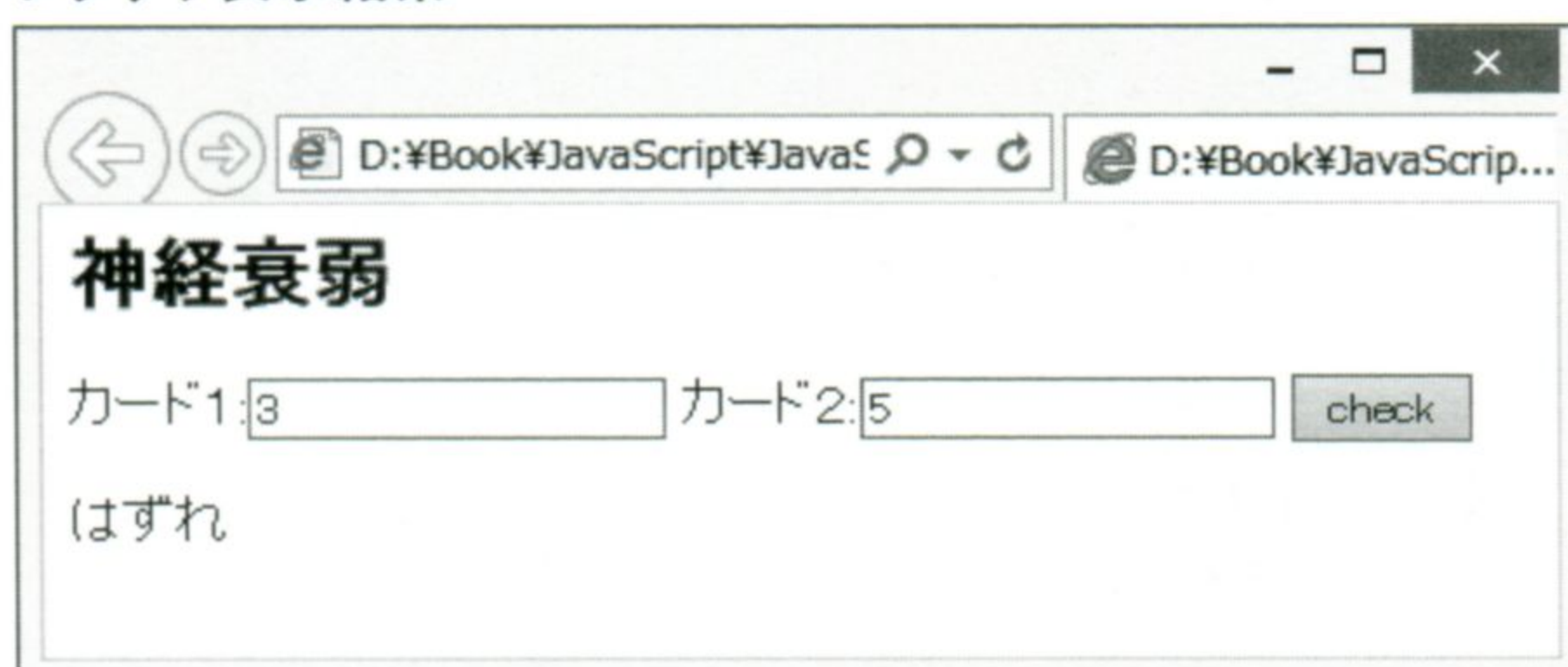
```
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            function check() {
                var a = document.getElementById("i0").value;
                var b = document.getElementById("i1").value;
                if (a == b) {
                    document.getElementById("result").textContent = "同じ！"
                }
                else {
                    document.getElementById("result").textContent = "はずれ"
                }
            }
        </script>
    </head>
    <body>
        <h2>神経衰弱</h2>
```

```
        カード1:<input id="i0" value="3" />
        カード2:<input id="i1" value="5" />
        <button onclick="check()">check</button>
        <p id="result"></p>
    </body>
</html>
```

ブラウザ表示結果

神経衰弱

カード1: 3 カード2: 5 check

はずれ

ボタンを押すとふたつのカードの比較結果が表示されます。同じ数字なら「同じ!」と、違う数字なら「はずれ」と表示されます。非常に原始的な例ですが、実際の神経衰弱ゲームでも同じような処理を行っています。

演習 **身長と体重から肥満度を判断する**

身長と体重を入力値として受け取り、BMIを計算し、その判定結果を表示するページを作ってください。BMIは「体重kg/(身長m)²」という計算式で求められ、18.5未満の場合には痩せすぎ、25以上は肥満気味、その中間は標準とされています。

SAMPLE BMI.html

```
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            function calcBMI() {
                // ここにコードを記述してください
            }
        </script>
    </head>
    <body>
        <h2>BMI計算機</h2>
        身長(m):<input id="i0" value="1.7" />
        体重(kg):<input id="i1" value="65" />
        <button onclick="calcBMI()">check</button>
        <p id="result"></p>
    </body>
</html>
```

3-3-3 | 複数の条件式を組み合わせる — ANDとOR

条件式をひとつ以上組み合わせる必要に迫られることもあるでしょう。

- 「残り1機 かつ 敵の弾が当たった」という条件のときにゲームオーバーにする
- 「スペード もしくは クローバ」という条件のときにカードをオープンする
- 「ある数値が100以上 かつ 200未満」のときに当たりとする

ゲームによっていろいろな状況が考えられるでしょう。個々の条件式は複数組み合わせることができます。

- AND条件式　　条件式1 && 条件式2 && 条件式3 …
- OR条件式　　条件式1 || 条件式2 || 条件式3 …

AND条件式は「かつ」です。すべての条件式がtrueのときに全体がtrueとなります。一方、OR条件式は「もしくは」です。個々の条件式のどれかがtrueのときに全体がtrueとなります。例を見てみましょう。

```
var a = true && true && true;       // a = true
var b = true && false && true;      // b = false
var c = true && true && false;      // c = false

var d = true || false || false;     // d = true
var e = false || true || false;     // e = true
var f = false || false || false;    // f = false
```

以下のようにANDとORを組み合わせることも可能です。

```
(条件式1 && 条件式2) || (条件式3 && 条件式4)
```

この例では、条件式1と条件式2がともにtrueか、もしくは、条件式3と条件式4がともにtrueのときに全体がtrueとなります。

AND条件とOR条件をテストするページを次に示します。

AND条件とOR条件をテストするページ SAMPLE andor.html

```
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            function update() {
                var a = document.getElementById("a").checked;
                var b = document.getElementById("b").checked;
                var c = document.getElementById("c").checked;
                document.getElementById("r0").textContent = (a && b && c);
                document.getElementById("r1").textContent = (a || b || c);
            }
        </script>
    </head>
    <body>
        <h2>AND/ORテスト</h2>
        A:<input id="a" type="checkbox" onchange="update()" />
        B:<input id="b" type="checkbox" onchange="update()" />
        C:<input id="c" type="checkbox" onchange="update()" />
        <p>
            A && B && C => <span id="r0"></span><br />
            A || B || C => <span id="r1"></span><br />
        </p>
    </body>
</html>
```

ブラウザ表示結果

D:¥Book¥JavaScript¥JavaS

AND/ORテスト

A: ☑ B: ☐ C: ☐

A && B && C => false
A || B || C => true

複数の条件式の判断

以下のような条件式があったとします。

```
if (条件式1  &&  条件式2  &&  条件式3  &&  条件式4) {
```

条件式が組み合わされた場合、左から右へ順番に評価が行われます。すなわち、条件式1→条件式2→条件式3→…と実行されます。ここで、条件式2がfalseだったとします。この時点で、全体の条件式はfalseと

なり、条件式3や4が評価されることはなくなります。なぜだかわかりますか? &&で複数の条件を組み合わせた場合、どれかひとつでもfalseだと全体がfalseになりしたよね。つまり、ひとつでもfalseがあると、その時点で全体の結果が決まってしまい、それ以降の条件式を実行することは無駄になるからです。一方、

```
if (条件式1 || 条件式2 || 条件式3 || 条件式4) {
```

という条件式の組み合わせがあったとします。この場合も条件式1→条件式2→…と左から評価が行われます。どれかひとつでも結果がtrueになった場合には、その時点で条件式の評価は中止され、条件式の組み合わせ結果はtrueとなります。

JavaScriptを見ていると以下のような記述をみかけることがあります。

```
var a = b || 3;
```

一見すると何がしたいのかわかりづらい記述ですが、条件式の挙動をうまく利用した一例です。bに何らか値が入っていれば、aにはbの値が代入されます。bに値があるということは条件式の評価ではtrueとみなされ、||より右は実行されません。

一方、bに値が何も入っていない場合、条件式bはfalseとみなされ、||より右の条件式の評価が行われます。この場合は、評価結果は3となり、aには3が代入されます。つまり、デフォルトの値を簡単に設定することができるのです。知っていると便利な小技のひとつです。

3-3-4 | 条件式 ー switch文

条件のパターンが多いときに、else ifを連続して使いたくなるかもしれません。

SAMPLE elseifelseif.html

```
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            window.onload = function () {
                var str = "";
                var day = new Date().getDay();  ←1
                if (day == 0) {
                    str = "日";
                } else if (day == 1) {
                    str = "月";
```

```
            } else if (day == 2) {
                str = "火";
            } else if (day == 3) {
                str = "水";
            } else if (day == 4) {
                str = "木";
            } else if (day == 5) {
                str = "金";
            } else if (day == 6) {
                str = "土";
            }
            document.getElementById("day").textContent = str;


        }
    </script>
</head>
<body>
    <h1>今日は<span id="day"></span>曜日</h1>
</body>
</html>
```

❶のgetDay()は曜日を返す関数です。「new Date().getDay()」で今日の曜日が取得できます。この程度の例であれば、else ifが連続しても見通しはそれほど悪くありませんが、条件式が複数行になってくると、一目では何をしているのかわかりにくくなります。

ある変数や式の取り得る値が複数パターンあり、それぞれに別の処理が必要な場合は、else ifではなくswitch文を使うことができます。switch文は以下のように記述します。

switch文の書式

```
switch (変数 もしくは 式) {   ←❶
    case 値1:       ┐
      命令文1;      ┘←❷
      break;       ←❸
    case 値2:       ┐
      命令文2;      ┘←❹
      break;       ←❺
    default:
      命令文;
      break;
}
```

switchの直後のかっこの中に変数もしくは式を記述します❶。その値が値1だったときに命令文1が❷、値

2だったときに命令文2が4…というようにパターンごとに適切な命令が実行されます。

case値の後ろは「;」（セミコロン）ではなく「：」（コロン）であることに注意してください。

それぞれのパターンの命令文の終わりにはbreakを挿入し35、処理がそこで終了する旨を明記します。breakを入れ忘れると、次のcaseも続けて実行されてしまいます。この動きを意図的に使う人もいますが、わかりにくいバグにつながるので、各caseの処理が終了したときには必ずbreakを挿入する習慣をつけたほうがよいでしょう。

値がどれにも該当しない場合もカバーしたい場合はcaseの代わりにdefaultを記述します。先ほどの例をswitchで書き換えた例を以下に示します。

SAMPLE switch.html

```
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            window.onload = function () {
                var str = "";
                var day = new Date().getDay();
                switch (day) {
                    case 0:
                        str = "日"; break;
                    case 1:
                        str = "月"; break;
                    case 2:
                        str = "火"; break;
                    case 3:
                        str = "水"; break;
                    case 4:
                        str = "木"; break;
                    case 5:
                        str = "金"; break;
                    case 6:
                        str = "土"; break;
                }
                document.getElementById("day").textContent = str;
            }
        </script>
    </head>
    <body>
        <h1>今日は<span id="day"></span>曜日</h1>
    </body>
</html>
```

若干見通しがよくなった気がしませんか?

演習 **西暦から干支（十干と十二支）を表示する**

以下に示すページを元に、西暦から干支（十干と十二支）を表示するページを作ってください。十干は西暦を10で割った余りからなり、余りが0から順に庚，辛，壬，癸，甲，乙，丙，丁，戊，己となります。十二支は西暦を12で割った余りからなり、余りが0から順に申，酉，戌，亥，子，丑，寅，卯，辰，巳，午，未となります。

SAMPLE switch-eto.html

```
<html>
    <head>
        <meta charset="UTF-8">
        <script>
                ...
            function calc() {
                var y = document.getElementById("year").value;
                ...
            }
        </script>
    </head>
    <body>
        <h2><span id="y"></span>年の干支は<span id="e"></span></h2>
        <p>
            <input id="year" value="2016"/>
            <button onclick="calc()">計算</button>
        </p>
    </body>
</html>
```

ブラウザ表示結果

C:¥Publication¥ゲ-
2016年の干支は丙申
2016 計算

C:¥Publication¥ゲ-
2020年の干支は庚子
2020 計算

3-3-5 | 条件式 ― 三項演算子

複雑なelse ifはswitchでシンプルになる場合もあることを見てきました。逆に「trueのときは値A、falseの時は値Bを代入する」というように非常に単純な条件分岐が必要になることもあります。もちろんif文を使っても記述できますが、JavaScriptを含む多くの言語では、単純な条件分岐をより簡潔に記述するために三項演算子というものが用意されています。次のように記述します。

```
条件式 ? 命令文1 : 命令文2
```

条件式がtrueのときに命令文1が実行され、falseのときに命令文2が実行されます。3つの要素（ひとつの条件とふたつの命令文）を「?」と「:」という演算子で関連付けるので「三項演算子」と呼ばれます。

例を見てみましょう。

```
document.getElementById("info").textContent = (v % 2 == 0) ? "even" : "odd";
```

変数vの値が偶数であれば画面に「even」と表示され、奇数であれば「odd」と表示されます。

%は余りを求める演算子です。2で割った余りが0なら条件式がtrueとなり、最初の命令文が実行されます。上の例では単なる「even」という文字列なので、その値が「info」というid属性を持つ要素のtextContentに代入されます。

if文を使ってこの例を記述すると2〜3行は必要になるでしょう。単純な条件処理であれば三項演算子を使うと簡便に記述できます。

ちなみに、「i++;」「i--;」のような演算子を「単項演算子」と呼びます。「+」「-」「*」「/」「%」といった演算子はふたつの数値の演算を行うので「二項演算子」と呼ばれます。

3-4 配列と繰り返し

ここでは、複数の値を一括して管理する配列と、それらを繰り返して処理する方法について学びます。配列はどのプログラミング言語でも基本となる要素です。しっかりマスターしましょう。

3-4-1 | 配列の使い方

同じ種類のものがたくさんある場合、それらを順番にならべると扱いやすくなります。実生活でも、出席番号順に生徒を整列させる、テストの平均点を計算するなど、さまざまな状況で行っている作業でしょう。プログラミングでも同じです。敵キャラクター、落ちてくるブロック、敵の弾（座標や速度･向き）、トランプカードなど、同じ種類のものがたくさんある場合に、それらを並べて管理します。

たとえば、5人の点数の平均を計算する必要があったとします。

```
var scoreA = 50, scoreB = 70, scoreC = 37, scoreD = 90, scoreF = 67;
var average = (scoreA + scoreB + scoreC + scoreD + scoreE) / 5;
```

5人位ならなんとかなります。しかし、これが100人になったらどうでしょうか？ 100個も変数を宣言するのは大変ですよね。ミスをする可能性も高くなります。修正も面倒です。

このような場合に、「配列」という機能を使用します。JavaScriptでは「[ ]」でくくられた部分が配列となります。配列の中のデータはカンマ区切りで並べます。データの種類は数値でも文字列でも何でもかまいません。

配列は箱の中に複数のデータを格納できる

配列の宣言

```
var point = [1, 5, 3, 6, 8, 2];
var names = ["tanaka", "suzuki", "kato", "sato"];
```

配列の中の個々の要素にアクセスするときは、変数名の後ろに「[先頭からの順番]」を指定します。この数値は0から始まることに注意してください。

```
var a = point[0];   // a = 1
var b = point[1];   // b = 5
var c = point[2];   // c = 3

var x = names[0];   // x = "tanaka"
var y = names[1];   // y = "suzuki"
var z = names[2];   // z = "kato"
```

配列に含まれている要素の個数は、「配列の変数名.length」とすると取得できます。

配列の要素数を取得

```
var p = point.length;   // p = 6
var q = names.length;   // q = 4
```

配列を使って先ほどの平均点の例を表現すると以下のようになります。

```
var scores = [50, 70, 37, 90, 67];
var average = (scores[0] + scores[1] + scores[2] + scores[3] + scores[4]) / scores.length;
```

沢山の変数を宣言しなくても済むようになりました。ただ、これだけだとありがたみは感じられません。しかし、次に説明する繰返し文と組み合わせると配列はその威力を発揮します。

演習　配列を定義してみよう

どんな内容でもかまいません。自分で配列を定義してみましょう。

SAMPLE array.html

```
var subjects = ["数学", "英語", "国語", "社会", "理科"];
var jikkan = ["庚", "辛", "壬", "癸", "甲", "乙", "丙", "丁", "戊", "己"];
var days = [ 31, 28, 31, 30, 31, 30 , 31, 31, 30, 31, 30, 31];
```

3-4-2 | 繰り返し ― for文

配列を使うと同じ種類のデータを並べて管理できます。何のために並べたのでしょうか? 順番に何らかの処理をするためです。でなければ配列にする意味がありません。ここではfor文の定義と、for文を使って配列の要素を順番に処理する方法をご紹介しましょう。「配列」と「for文」の組み合わせ、これは「条件式」と「if文」の組み合わせに優るとも劣らない、プログラミングにおける最強タッグといっても過言ではありません。あらゆるところで使用されるテクニックなのでぜひとも慣れるようにしてください。

for文はある条件下で処理を繰り返すときに使用します。英語のループ（loop）とは輪を意味しますが、処理を繰り返すことを「ループ」と呼びます。

for文の書式

```
        1              2                  4
for (初期化式 ; ループ継続の条件式 ; カウンタ変数の更新) {
    命令文  ←3
}
```

for文は以下のような順序で実行します。

1. 初期化式を実行
2. ループ継続の条件式がtrueであればステップ3へ、falseであればfor文終了
3. 命令文の実行
4. カウンタ変数の更新
5. ステップ2に戻る

for文を使って先ほどの点数の平均を求める部分を書き直すと以下のようになります。

```
var scores = [50, 70, 37, 90, 67];
var total = 0, average;
             1           2 5          4
for (var i = 0 ; i < scores.length ; i++) {
    total += scores[i]; ←3
}
average = total / scores.length;
```

ループ1回目は次のようになります。

1. for文でカウンタ変数iを0で初期化
2. 配列のlengthプロパティで配列に含まれる個数を取得。scoresには5個データが含まれているので、scores.lengthは5となる。「i<scores.length」は「0<5」なので条件式はtrue、よって、命令文が実行される

3 命令文「total += scores[i]」はiが0なので、「total += scores[0]」となる。scoresの先頭の値は50なので、変数の値は「total = 0 + 50 = 50」となる

4 i++を実行することでiの値が1増加して0から1になる

5 ステップ**2**に戻り、条件式の評価から処理が継続される

ループ2回目以降の各値は以下のようになります。

ループ2回目以降の値

ループ	i	条件式	命令文（totalの値）
2回目	1	1＜5はtrue	total += scores[1]で total＝50＋70＝120
3回目	2	2＜5はtrue	total += scores[2]で total＝120+ 37＝157
4回目	3	3＜5はtrue	total += scores[3]で total＝157+ 90＝247
5回目	4	4＜5はtrue	total += scores[4]で total＝247+ 67＝314
6回目	5	5＜5はfalse	命令文は実行されず、for文の次へ

演習　1から指定した数までの合計を求める

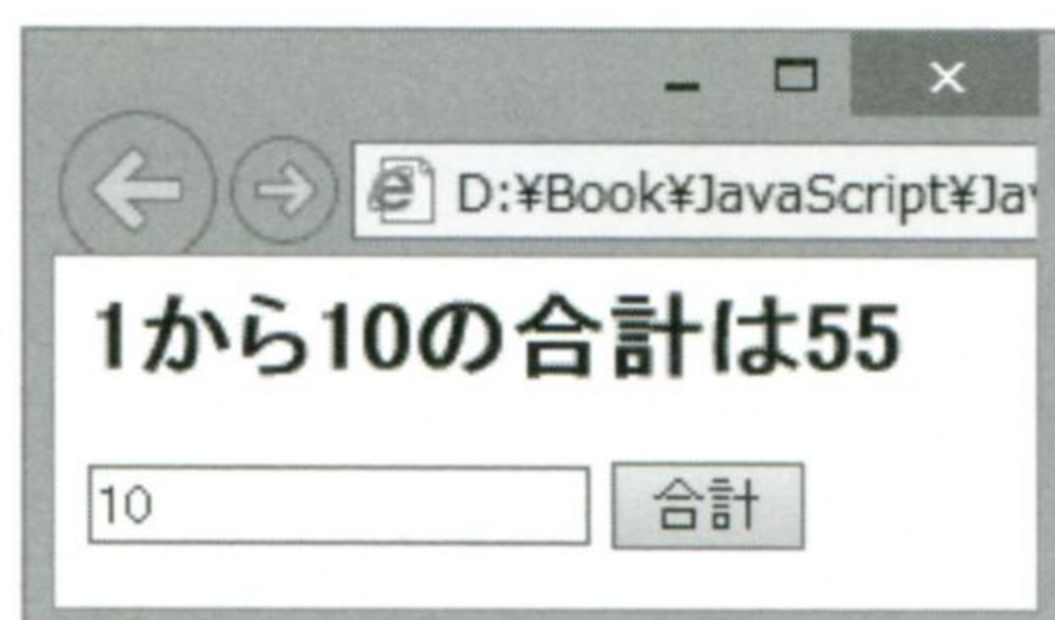
1からNまでの数値の合計を求めるページを作ってみましょう。以下のコメント部分にコードを記入してください。

SAMPLE　1-to-N.html

```
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            function calcSum() {
                var max = document.getElementById("max").value;
// <ここにコードを記述してください>
                document.getElementById("val").textContent = max;
                document.getElementById("sum").textContent = total;
            }
        </script>
    </head>
    <body>
        <h2>1から<span id="val"></span>の合計は<span id="sum"></span></h2>
        <input id="max" value="10" />
        <button onclick="calcSum()">合計</button>
    </body>
</html>
```

D:¥Book¥JavaScript¥Ja

1から10の合計は55

10　合計

3-4-3 | 繰り返し − while文

while文はfor文とよく似ています。ループ継続の条件式がtrueである間、命令文の実行を繰り返します。

while文の書式

```
while (ループ継続の条件式) {
    命令文
}
```

演習 **前述の演習のプログラムをwhile文を使って書き直してみよう**

while文を使って先ほどの演習の1からNまでの数値の合計を求めるページを書き直してみましょう。

SAMPLE 1-to-N-while.html

ちなみにfor文の初期化式とカウンタ更新を省略するとwhile文とまったく同じ動きになります。なので、while文はfor文で代用することも可能です。

以下は同じ

```
while (ループ継続の条件式) {
  …
}
```

```
for (;ループ継続の条件式;) {
  …
}
```

3-4-4 | 繰り返し − continue文、break文

for文やwhile文といったループ処理において、その流れを変えるための命令です。

while文の場合

```
while (条件式1) {
    ...
    if (条件式2) {
        continue;     // 条件式1へ
    }
    ...
    if (条件式3) {
        break;        // ループを抜ける
    }
    ...
}
```

while文でcontinueが呼ばれると、continue文以降の処理は実行されることなく、「条件式1」の評価に戻り、while文の処理が継続されます。

break文が呼び出されると、ループの処理が終わり、制御がwhile文の次に移動します。

for文の場合

```
for (初期化式; 条件式1; カウンタ更新) {
    ...
    if (条件式2) {
        continue;    // カウンタ更新へ
    }
    ...
    if (条件式3) {
        break;       // ループを抜ける
    }
    ...
}
```

一方、for文の場合は、continueが呼ばれると、continue文以降の処理は実行されることなく、カウンタ更新に処理が移動し、そのあと条件式1の評価が行われ、for文の処理が継続されます。break文が呼び出されると、ループの処理が終わり、制御がfor文の次に移動します。このようにcontinue～breakを使うと、ループ処理の中で細かい制御ができるようになります。

このほかにもdo～while文などを使って繰り返し処理の制御を行うことができますが、当面はこれだけ覚えていれば十分でしょう。

演習 for文で回文を表示するページをつくろう

for文を使って回文を表示するページをつくりましょう。文字列の長さは「文字列変数.length」で、文字列のN番目の文字は「文字列.charAt(N)」で取得できます。

SAMPLE kaibun.html

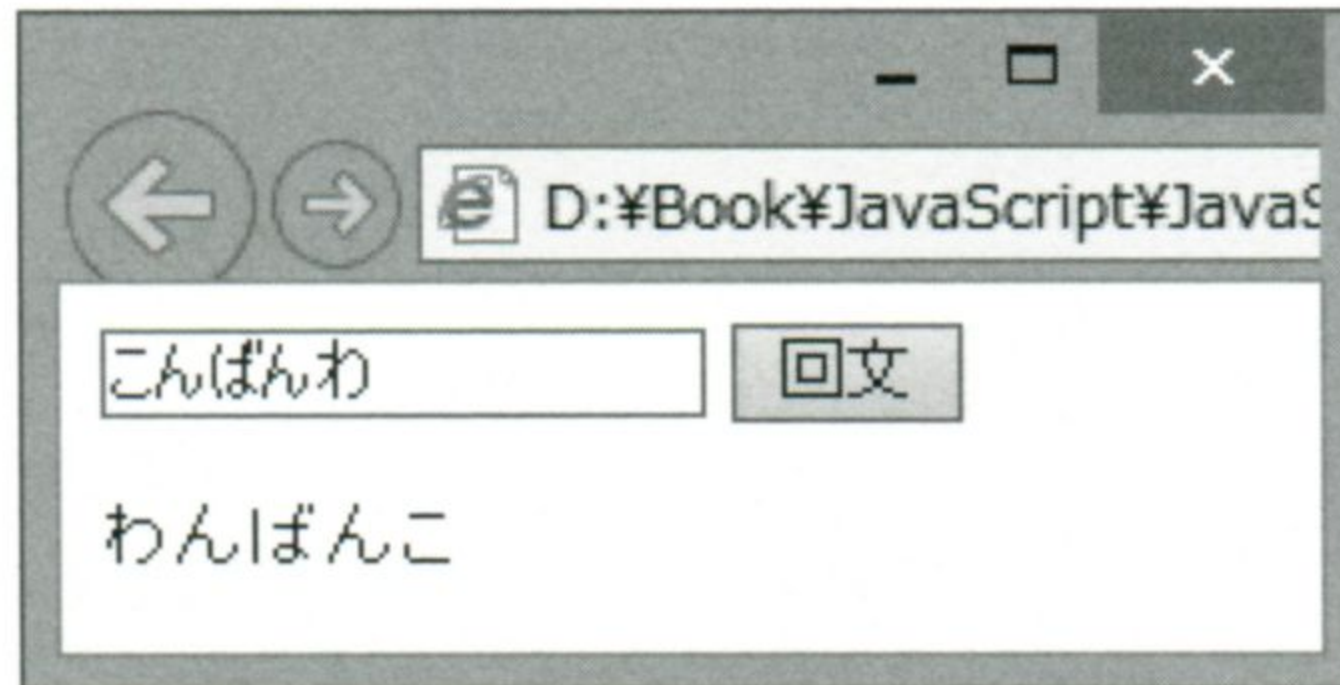
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            function kaibun() {
                // ここにコードを記述してください
            }
        </script>
    </head>
    <body>
        <input id="source"/>
```

```
        <button onclick="kaibun()">回文</button>
        <p id="result"></p>
    </body>
</html>
```

ブラウザ表示例

D:¥Book¥JavaScript¥JavaS

こんばんわ 回文

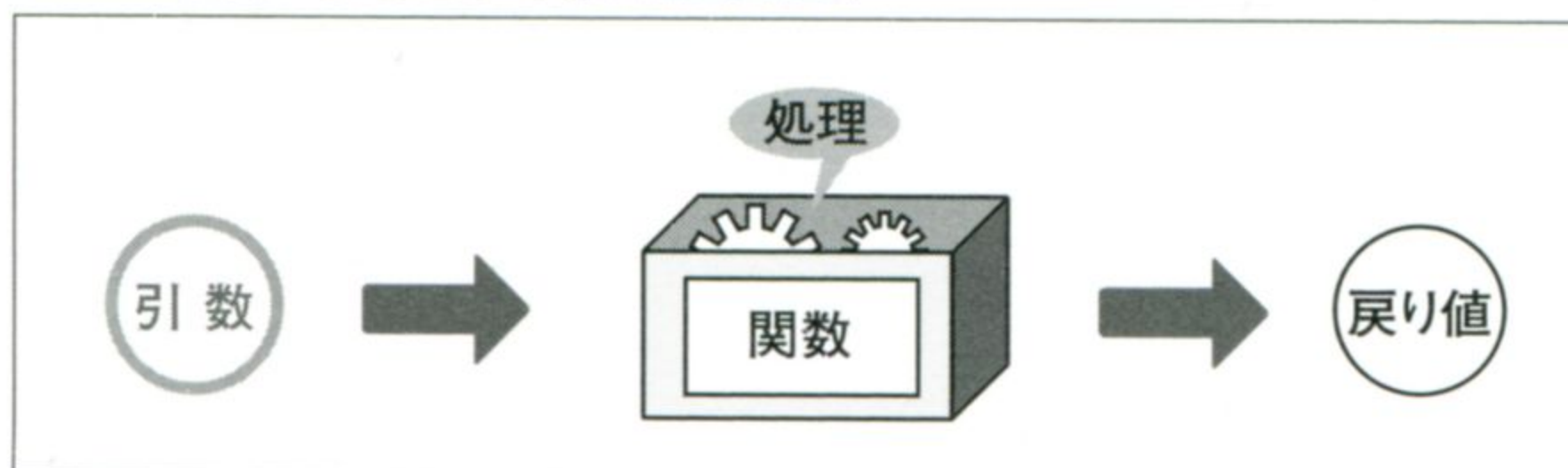
わんばんこ

3-5 関数

関数とは「ある程度まとまった処理をひとつの機能として抽象化する」という働きをします。プログラミングの発展の歴史は、抽象化を進めてきた歴史といってもよいでしょう。関数は最初のレベルの抽象化にあたります。

3-5-1 関数の定義

中学数学で、一次関数「y=ax+b」、二次関数「y =ax^2+bx+c」というように関数について学習したと思います。関数は変数xに値を入れるとyの値が決まるので、「y=f(x)」とも表記されます。このように、関数とは、入力値を受け取って、何らかの処理を行い、その結果を出力として返すものでした。プログラミングにおける関数も同じです。関数とは、引数と呼ばれる入力値を受け取って、何らかの処理を行い、その結果を戻り値として返すものです。

関数は引数を処理して戻り値を返す

処理
引数 → 関数 → 戻り値

JavaScriptでは以下のようにfunctionというキーワードを使って関数を定義します。戻り値を返すときにはreturn文を使います。戻り値が不要なときはreturn 文を省略できます。

関数の定義

```
function 関数名(引数1, 引数2, …) {
    何らかの処理
    return 戻り値;
}
```

関数は一度定義してしまえば、何度でも気軽に呼び出せます。内部構造や動作原理を理解していなくても利用できる装置や機構のことを「ブラックボックス」といいますが、関数もまさにブラックボックスと同じです。

簡単な例を示しましょう。ふたつの数値を引数として受け取り、それらを足した結果を返す関数addは次のようになります。

```
function add(a, b) {
    return a + b;
}

var f = add(2, 5);        // f = 7
```

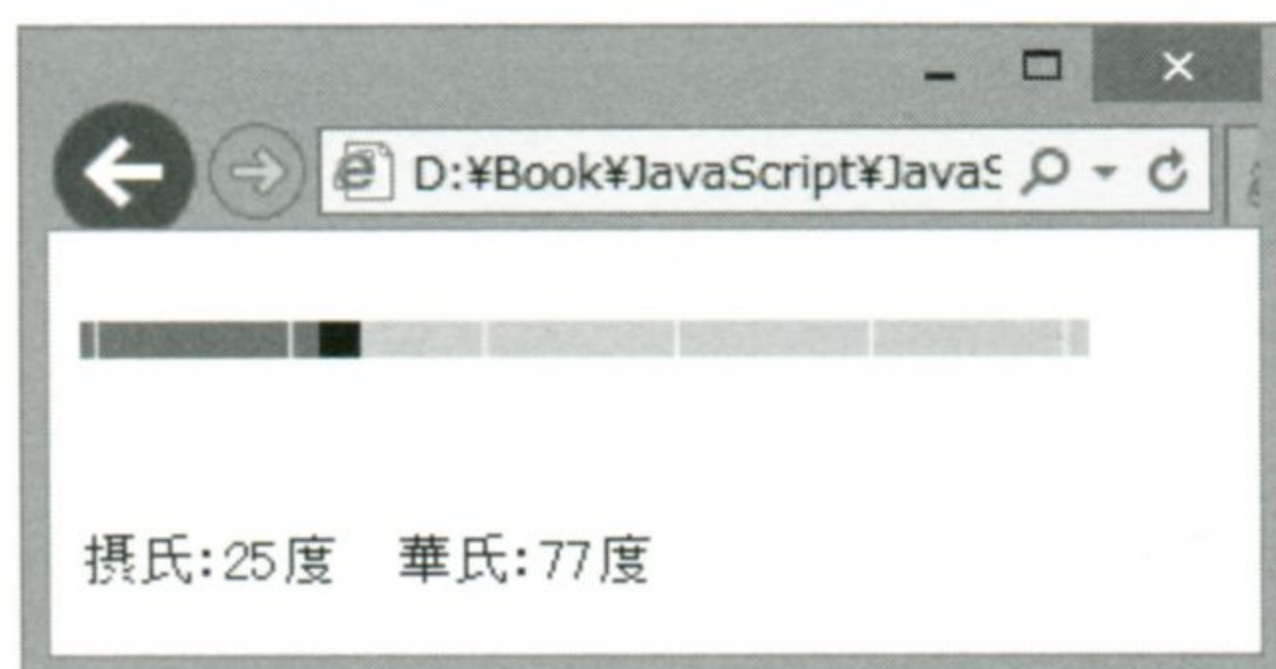
関数を呼び出す場合は、関数名の直後に「(引数1, 引数2,…)」というように、カッコの中に引数を指定します。引数の個数は自由に指定できますが、呼び出される側が意図している引数を設定しましょう。

たとえば、上の例にある関数addは2個の引数が渡されることを前提としています。したがって、呼び出し側も2個の引数を指定してください。引数を0個、1個、3個と違った個数で呼び出すと、正しい足し算の結果は取得できません。関数が正しく動作するためには適切な引数が必要になりますが、適切な引数を指定するのは呼び出し側の責任です。ちなみに、関数によっては引数がないものや戻り値がないものもあります。

演習 摂氏を華氏に変換する関数を定義してみよう

摂氏(celsius)から華氏(fahrenheit)に変換する関数c2fを定義してください。計算結果が小数点になるかもしれません。「Math.floor(小数点を含む任意の数値)」を使うと、小数点以下を切り捨てた整数値を得ることができます。

D:¥Book¥JavaScript¥JavaS

摂氏:25度　華氏:77度

SAMPLE c2f.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            function c2f(c) {
                // ここにコードを記述してください
            }

            function convert() {
                var c = document.getElementById("celsius").value;
                var f = c2f(c);
                var s = "摂氏：" + c + "度　華氏：" + f + "度";
                document.getElementById("result").textContent = s;
            }
```

```
        </script>
    </head>
    <body>
        <input id="celsius" type="range" min="0" max="100"
onchange="convert()"/>
        <p id="result"></p>
    </body>
</html>
```

▶変数のスコープ

変数が使える範囲を「スコープ」と呼びます。変数はどこで宣言するかによって、使用できる範囲が変わってきます。関数の中で宣言した変数は関数の中からしか使用できません。一方、関数の外で宣言した変数は、どこからでも使用することができます。このような変数を「広域変数」もしくは「グローバル変数」と呼びます。

変数のスコープ SAMPLE scope.html

```
<script>
    var score = 0, timer;
    function keydown(e) {
        var code = e.keyCode;
        if (code == 34) {
            // ...
        }
        score += 100;
        // ...
    }

    function tick() {
        var now = new Date();
        if (score > 1000) {
            clearInterval(timer);
        }
        // ...
    }
</script>
```


変数scoreとtimerは関数の外で宣言されています。つまり広域変数です。よって、どこからで参照することができます。上の例でも、keydownの中でscoreを、tickの中でscoreとtimerを参照しています。一方、変数codeはkeydownの中で宣言されているので、関数keydownの中でしか使うことができません。同様に、nowはtickの中でしか使えません。

広域変数はどこからでも使えるので便利なのですが、プログラムの規模が大きくなってくると、どこで何が行われるのか把握するのが困難になってしまいます。広域変数の使用はできるだけ控え、主な作業はローカル変数ですませるのが堅牢なプログラムを作成するコツのひとつです。

3-6 プログラムのバグをとる作業デバッグ

予想したとおりにプログラムが動けばよいのですが、現実はそれほど甘くありません。一般的に、デバッグとは問題を特定して修正する作業を意味しますが、他人の書いたコードを理解する場合にもとても有用なスキルとなります。

3-6-1 | ブラウザのデバッガー

「デバッグ」とはプログラムの不具合を特定して、その問題を修正することです。デバッグをするためのツールを「デバッガー」と呼びます。最初に書いたプログラムがそのまま問題なく動作するということはまずありません。開発をしているとほぼ例外なくデバッグというプロセスが必要になります。デバッガーを使うとプログラムを1行ずつ実行できるので、不具合の特定はもちろん、プログラムの動作理解にもとても役に立ちます。デバッガーを制するものはプログラミングを制するといっても過言ではありません。そのくらいデバッガーの習得は重要なスキルです。ぜひ、この機会にデバッガーに慣れるようにしてください。

最近のブラウザには開発ツールがついており、そのなかからデバッガーを起動することができます。ブラウザのバージョンアップサイクルは早いため、本書の画面と実際の様子は異なるかもしれません。しかし、デバッグの基本的な操作手順は変わりません。以下は多くのブラウザに備わっている主なデバッグ機能です。

- **ブレークポイントの設定・解除**
  プログラムを一時停止させる場所を「ブレークポイント」と呼びます。その場所を設定したり、解除したりする機能です。
- **ステップ実行**
  1行ずつ実行していく機能です。
  ステップイン：次の行を実行します。次の行が関数の場合、その関数の最初の行へ制御を移します。
  ステップオーバー：次の行を実行します。次の行が関数の場合、その関数を実行したあとで、関数の外で止まります。
  ステップアウト：現在の関数を最後まで実行し、呼び出し元の関数に戻ります。
- **変数の確認**
  変数にどんな値が格納されているか確認する機能です。広域変数とローカル変数を整理して表示したり、自分で変数を指定したりすることができます。「ウォッチ」機能とも呼ばれます。
- **コールスタックの確認**
  ある関数Aが関数Bを呼び出し、関数Bが関数Cを呼び出し、と関数呼び出しがどんどん深くなっていくことがあります。その関数呼び出しの履歴をコールスタックと呼びます。「呼び出し履歴」と翻訳されているかもしれません。

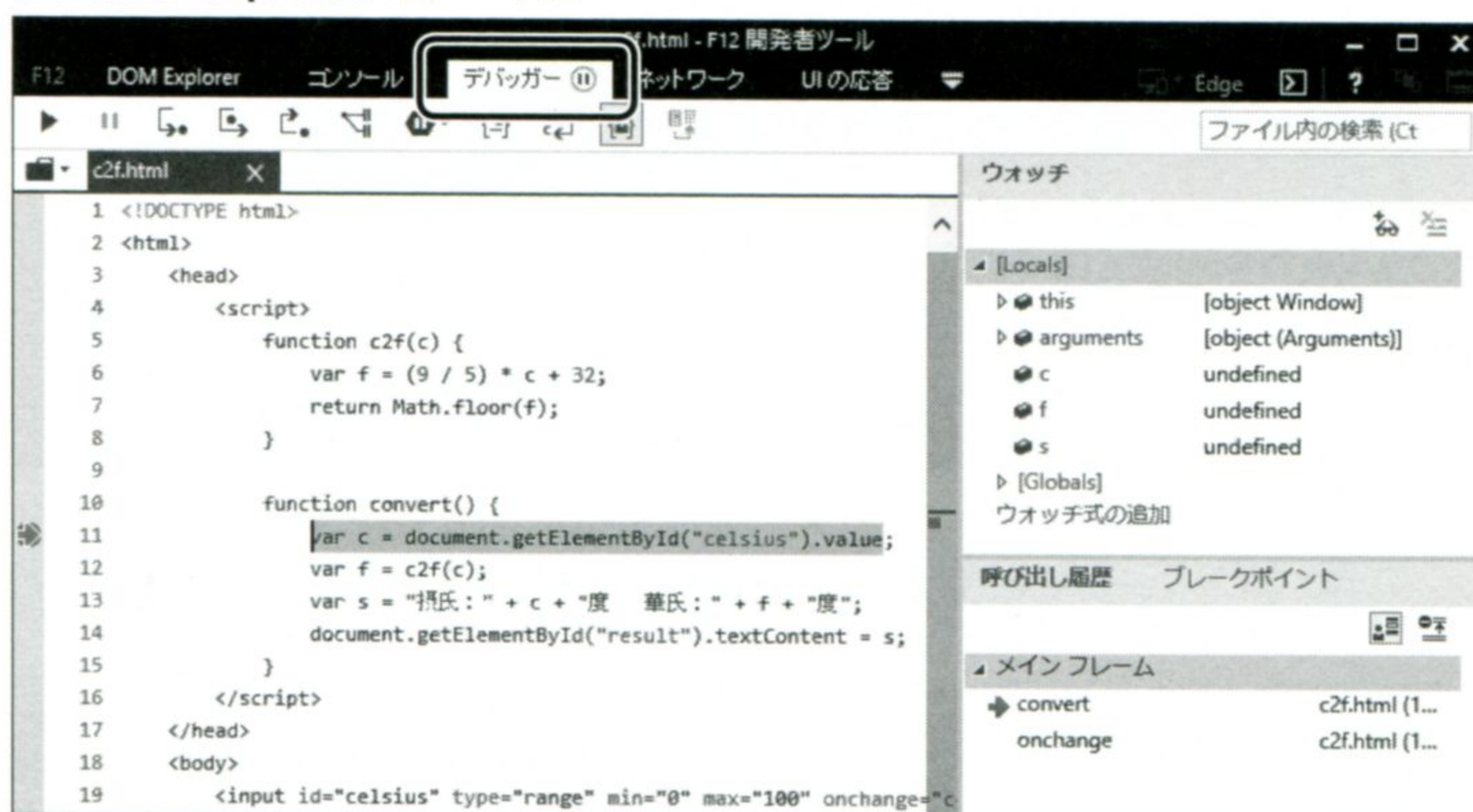
例として、P.091の演習でとりあげた、摂氏から華氏に変換するページでデバッガーを使ってみましょう。

摂氏から華氏に変換（P.091） SAMPLE c2f.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            function c2f(c) {
                var f = (9 / 5) * c + 32;
                return Math.floor(f);
            }

            function convert() {
                var c = document.getElementById("celsius").value;
                var f = c2f(c);
                var s = "摂氏：" + c + "度　華氏：" + f + "度";
                document.getElementById("result").textContent = s;
            }
        </script>
    </head>
    <body>
        <input id="celsius" type="range" min="0" max="100" onchange="convert()"/>
        <p id="result"></p>
    </body>
</html>
```

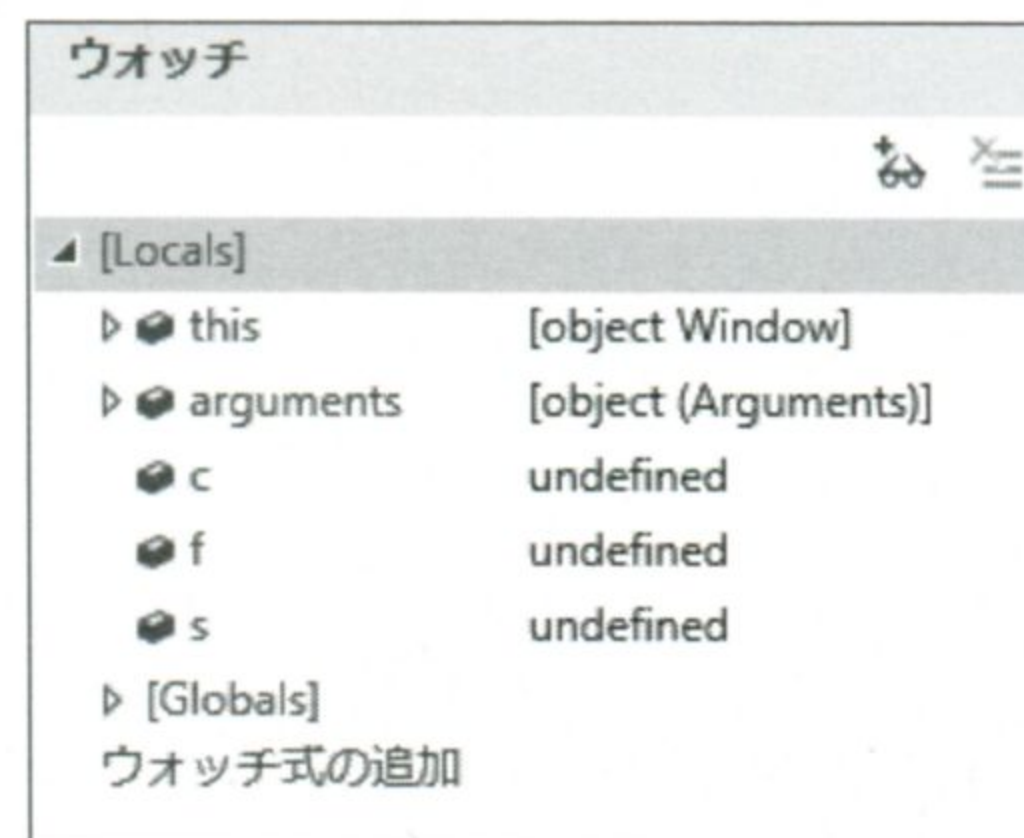
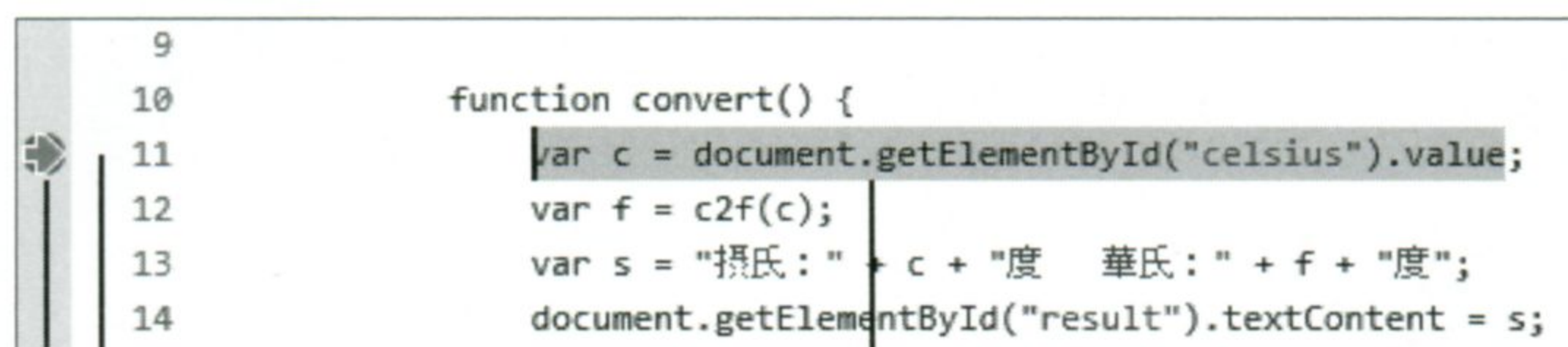
▶Internet Explorerの場合

ページを表示中にF12キーを押下するとデバッグツールが起動します。「デバッガー」タブを選び、convert関数1行目の行番号の左の領域をクリック、もしくはF9キーを押下して、関数にブレークポイントを設定します（次ページ図参照）。正しく設定されると赤い丸が表示されます。その状態でページ上のスライダバーを操作すると関数が呼び出され、ブレークポイントで実行が一時停止します。

Internet Explorerのデバッガー

現在実行が停止されている行が黄色でハイライトして表示されます。画面右のウォッチウインドウで、ローカル変数や広域変数、そのほか任意の変数の値を見ることができます。

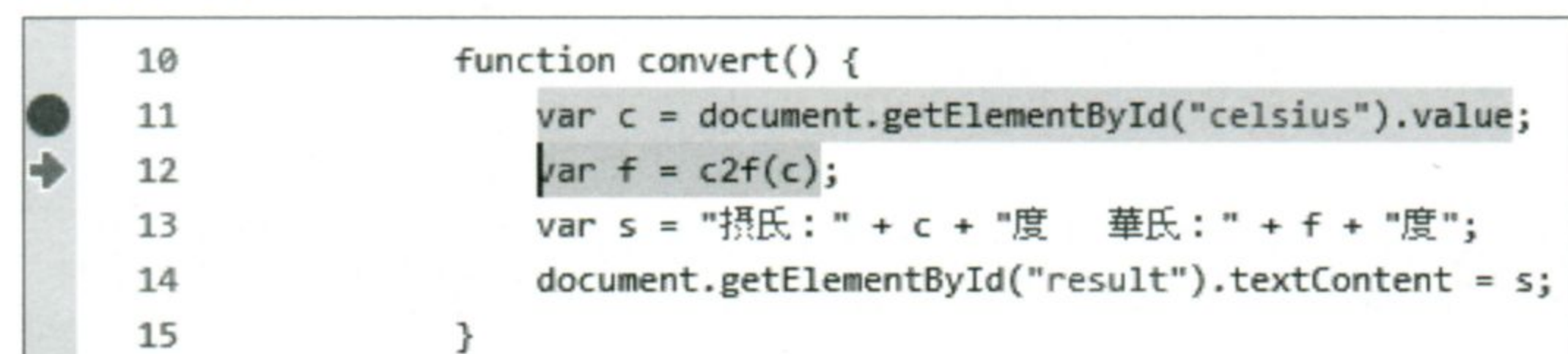
ここでツールバーにある3つのボタンを使って処理を進めていきます。ブラウザのバージョンによってアイコンが異なりますが、どこかにこの3つの機能をもつボタンがあるはずです。

ブレークポイント、ステップ実行ボタン、変数の値の表示

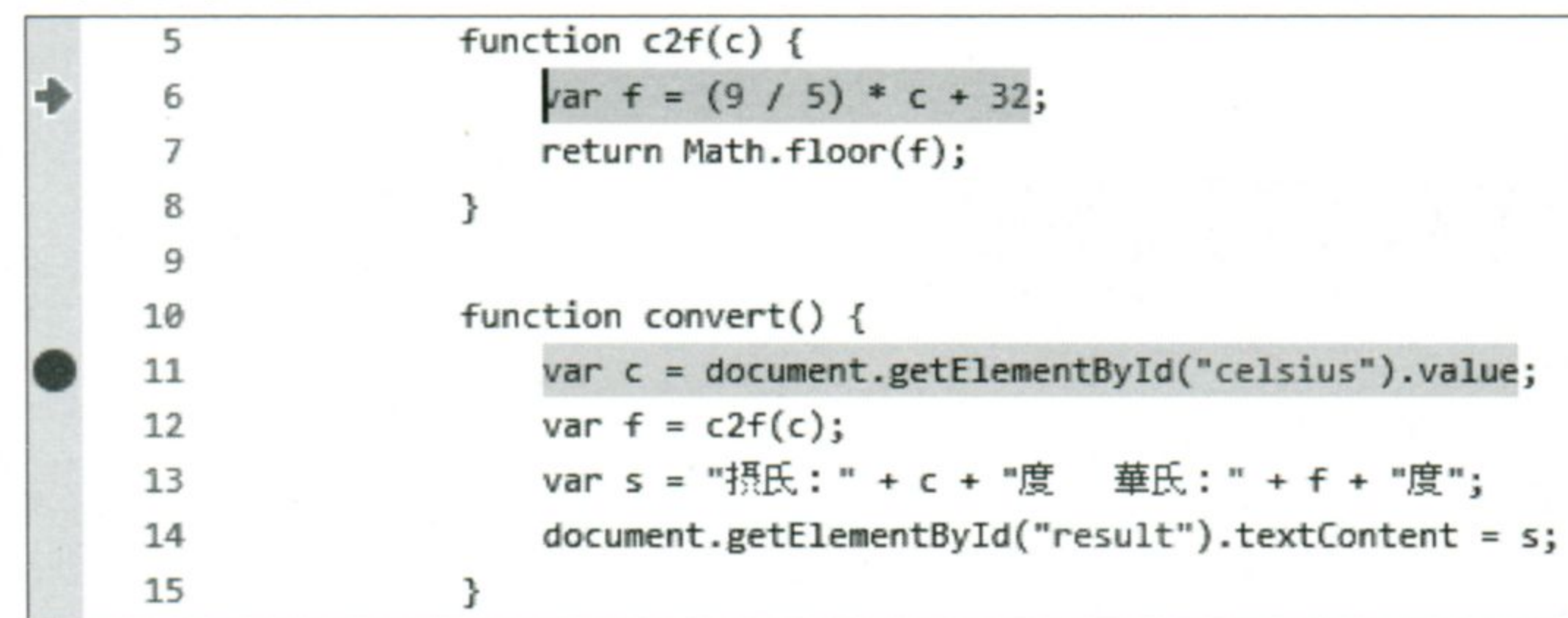

```
 9
10            function convert() {
11                var c = document.getElementById("celsius").value;
12                var f = c2f(c);
13                var s = "摂氏：" + c + "度　華氏：" + f + "度";
14                document.getElementById("result").textContent = s;
```

画面右側のウォッチウインドウでは変数の値を見ることができる

上記の状態でステップインアイコンをクリック（もしくはF11を押下）すると、以下のように「var f = c2f(c);」の行にフォーカスが移動します。

ステップイン1

```
10            function convert() {
11                var c = document.getElementById("celsius").value;
12                var f = c2f(c);
13                var s = "摂氏：" + c + "度　華氏：" + f + "度";
14                document.getElementById("result").textContent = s;
15            }
```

このあと、さらにステップインを実行すると、次のようになります。

ステップイン2

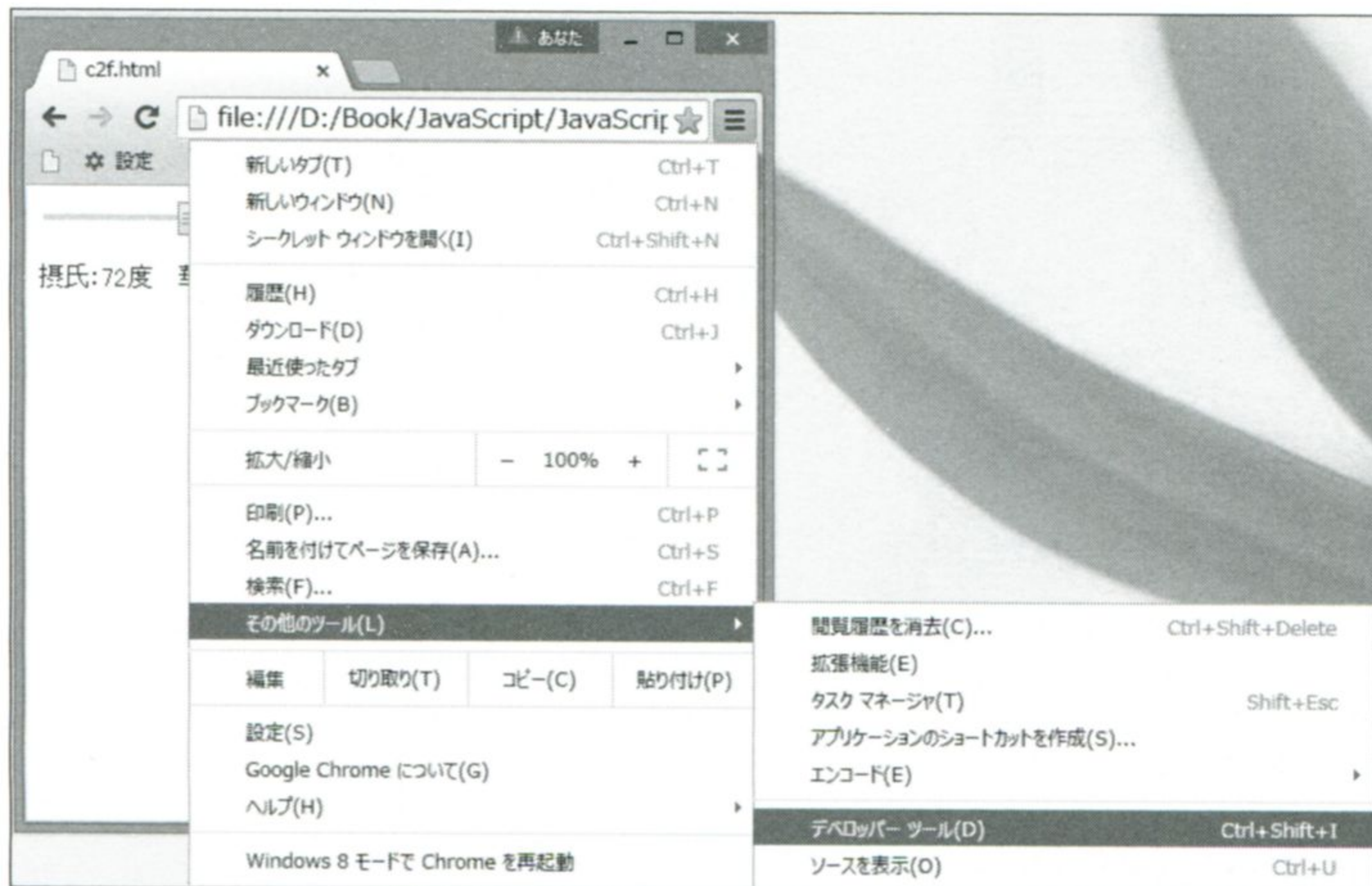
```
 5            function c2f(c) {
 6                var f = (9 / 5) * c + 32;
 7                return Math.floor(f);
 8            }
 9
10            function convert() {
11                var c = document.getElementById("celsius").value;
12                var f = c2f(c);
13                var s = "摂氏：" + c + "度　華氏：" + f + "度";
14                document.getElementById("result").textContent = s;
15            }
```

c2f関数の中に制御が移動していることがわかります。その後、ステップアウト、ステップオーバーと実行すると以下のようになります。

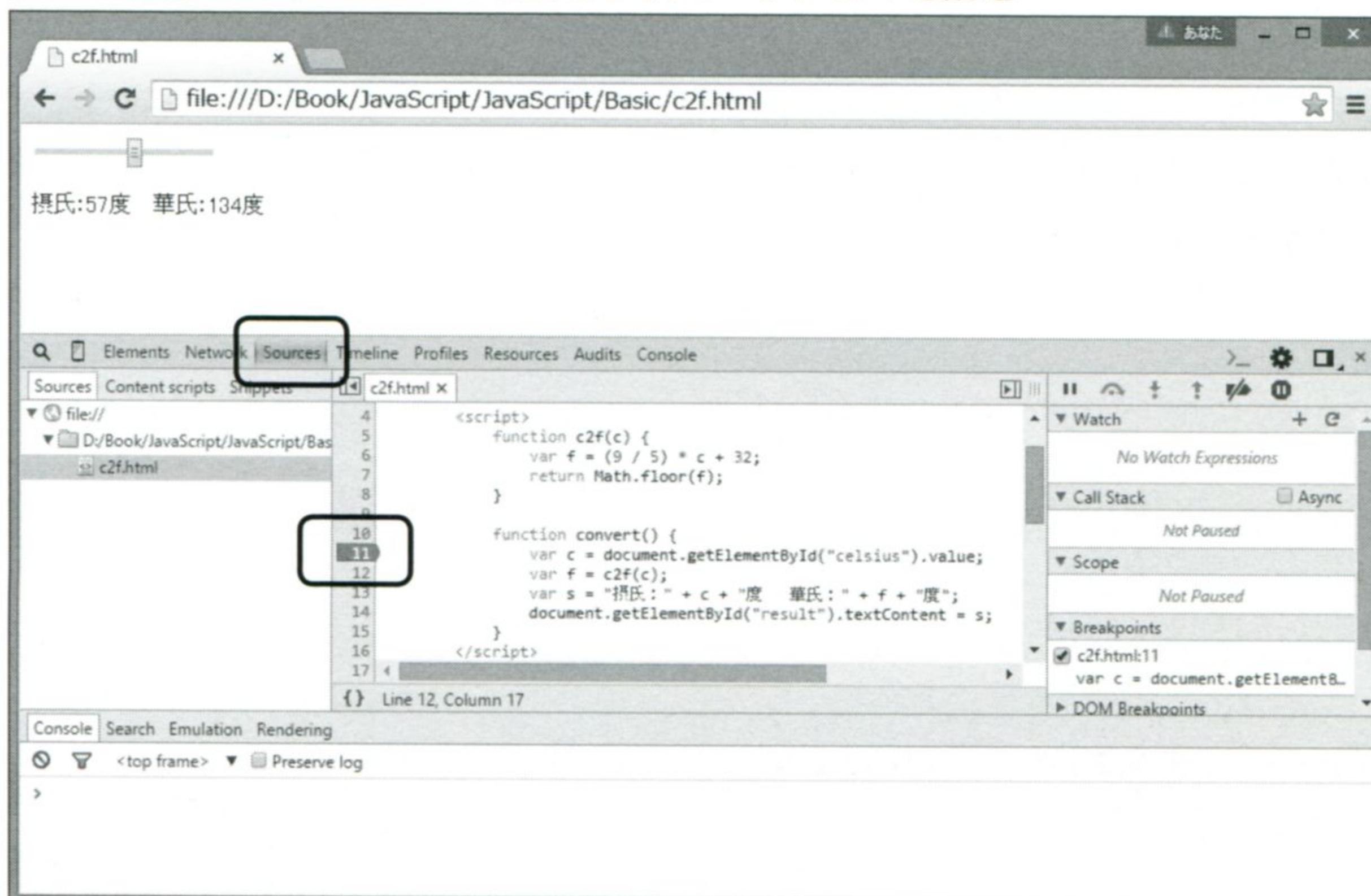
ステップアウト、ステップオーバー

```
10        function convert() {
11            var c = document.getElementById("celsius").value;
12            var f = c2f(c);
13            var s = "摂氏：" + c + "度　華氏：" + f + "度";
14            document.getElementById("result").textContent = s;
15        }
```

▶Google Chromeの場合

「Google Chromeの設定」→「その他のツール」→「デベロッパーツール」で、デベロッパーツールを起動します。

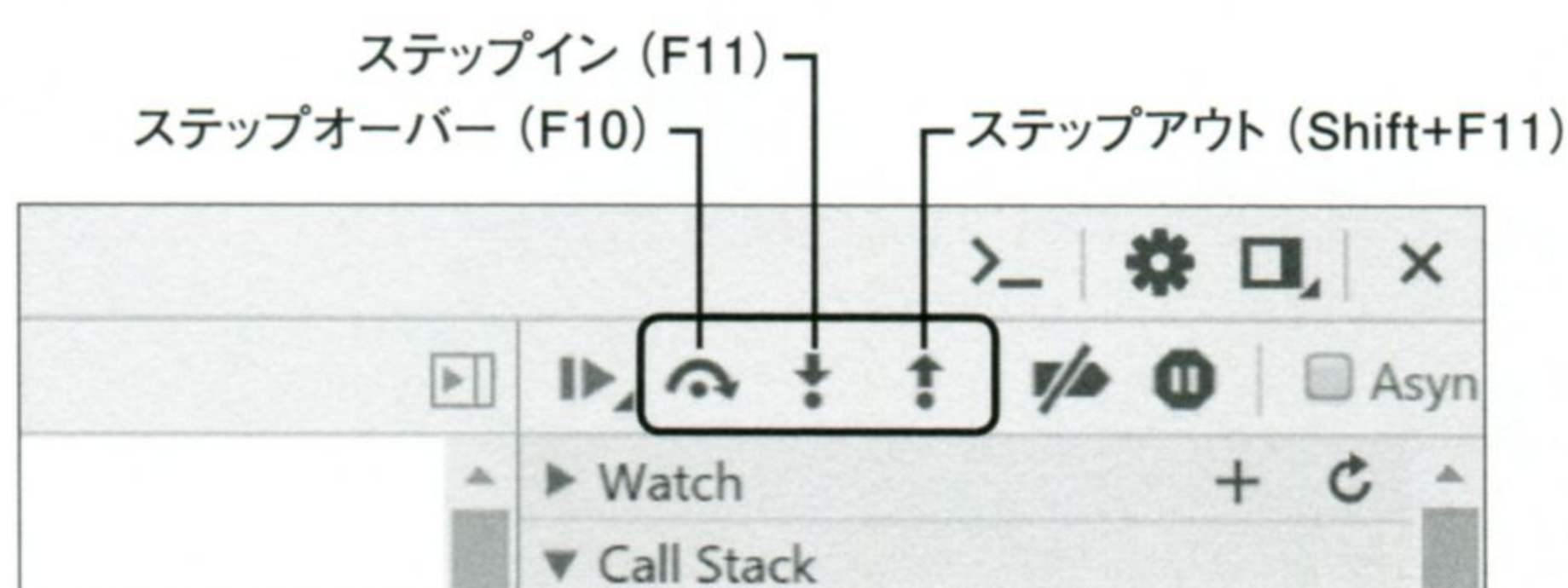
デベロッパーツールを起動

デベロッパーツールの表示状態は使用状況によって異なります。「Sources」タブを選択し、左側の一覧からHTMLファイルを選択してソースコードを表示し、行番号をクリックすることでブレークポイントが設定されます。

「Sources」タブでソースコードを表示してブレークポイントを設定

関数の1行目にブレークポイントを設定し、ページのスライダバーを動かします。現在止まっている箇所が青色でハイライトされます。

ブレークポイントで一時停止

```
10        function convert() {
11            var c = document.getElementById("celsius").value;
12            var f = c2f(c);
13            var s = "摂氏：" + c + "度　華氏：" + f + "度";
14            document.getElementById("result").textContent = s;
15        }
```

上記の状態でステップオーバーアイコンをクリック（もしくはF10を押下）すると、以下のように「var f = c2f(c);」の行にフォーカスが移動します。

ステップ実行ボタン

ステップオーバー

```
10        function convert() {
11            var c = document.getElementById("celsius").value;  c
12            var f = c2f(c);
13            var s = "摂氏：" + c + "度　華氏：" + f + "度";
14            document.getElementById("result").textContent = s;
15        }
```

続けて、ステップインを実行すると、次のようになります。

ステップイン

```
5         function c2f(c) {  c = "29"
6             var f = (9 / 5) * c + 32;  f = undefined
7             return Math.floor(f);
8         }
9
10        function convert() {
11            var c = document.getElementById("celsius").value;  c
12            var f = c2f(c);
13            var s = "摂氏：" + c + "度　華氏：" + f + "度";
14            document.getElementById("result").textContent = s;
15        }
```

c2f関数の中に制御が移動していることがわかります。その後、ステップアウト、ステップオーバーと実行すると以下のようになります。

```
10        function convert() {
11            var c = document.getElementById("celsius").value;  c
12            var f = c2f(c);  f = 84
13            var s = "摂氏：" + c + "度　華氏：" + f + "度";  s =
14            document.getElementById("result").textContent = s;
15        }
```

演習　デバッガーを使ってみよう

今まで入力したプログラムでデバッガーを使ってみましょう。いろいろな場所にブレークポイントを設定し、ステップイン、ステップアウト、ステップオーバーでどのように動くか確認してください。その際に変数の値（ローカル変数、広域変数）がどうなるか、コールスタック（呼び出し履歴）がどうなっているかも合わせて確認してください。

▶動かないときは

正しく入力したはずなのに動かない …… おそらく多くの人はそのような状況に遭遇することでしょう。詳しい人に聞くのもよいですが、それは最後の手段として取っておきましょう。試行錯誤する過程で多くのことを学ぶことができるからです。まずは自分である程度解決する努力をすることが大切です。

- 何度も見なおす
  当然のことですが、自分の書いたコードと本に掲載されているコードを丁寧に見比べてください。きっとどこかに間違いがあるはずです。
- デバッグをする
  開発者ツールを使って、どこに問題があるか調べてみましょう。コンソールに何らかのエラーが表示されている可能性も高いので、それにも注意します。いろいろなところにブレークポイントを設定し、どこで何が起きているのか把握することは非常に良い勉強になるはずです。
- バグを確実に再現する手順を見つける
  何らかのきっかけで動作がおかしくなることもあるでしょう。そんなときはその不具合を確実に再現できる手順をみつけるとバグの特定が早くなります。その手順に該当する箇所にブレークポイントを設定し、何が起きているか理解しながら実行していくとよいでしょう。
- コンソールに情報を出力してみる
  タイミングの問題でデバッガーを使うとバグが再現しないということもあるでしょう。そのような場合は、console.log()を使って各種情報をコンソールに出力し、何が起きているか調べてみましょう。
- ほかのブラウザ、OSで試してみる
  HTML5で標準化が進んだとはいえ、まだまだブラウザやOSによって挙動が異なる場合があります。特に新しい機能では挙動が異なることが少なくありません。もしかするとあなたのプログラムではなく、ブラウザの挙動の違いが原因なのかもしれません。行き詰まったときはほかの環境で試してみましょう。

よくあるミスを以下に挙げましょう。

- スペルが違う
  実際には一番多いケースのひとつです。以下は実際にあったケースです。何が問題かすぐにわかりますか?

```
ver a = 3;  // NG
var a = 3;  // OK
document.getElementById("info").textContent = "hello"; // OK
document.getElementByid("info").textContent = "hello"; // NG
```

よく見るとvarがverになっている、Idの「i」が小文字になっている、という間違いがあることがわかります。「このくらいのミスは見逃してくれよ」と思われるかもしれませんが、コンピュータはどんな些細なミスも許してくれないのでご注意ください。

- カッコの対応がとれていない

これもよくあるケースです。デバッガーでブレークポイントを設定できない場合、これが原因のことがほとんどです。カッコの対応に応じてきちんとソースコードの体裁を整えてくれる統合開発環境を使うことで、このミスを激減することができます。以下は誤った例です。

```
function init() {
    for (var i = 0 ; i < 10 ; i++) {
        document.getElementById("id"+i).textContent = "";
}
```

関数が定義されているように見えますが、これは関数として不正です。なぜならカッコの対応がとれていないからです。

```
function init() {   ← これに対応するかっこ「}」がない
    for (var i = 0 ; i < 10 ; i++) {
        document.getElementById("id"+i).textContent = "";
    }
```

- 全角文字を使っている

非常に見つけにくいバグです。セミコロンが半角の「;」ではなく、日本語の「；」だったり、全角スペースが紛れていたこともありました。この手のバグは目視だけでは見つけることが困難です。ブラウザのデバッガーを使って不具合の箇所を特定する必要があります。
頭を掻きむしりながら「なぜ動かないんだ!」とイライラすることもあるでしょう。「PCが悪いにちがいない」と思うこともあるでしょう。「何でこんなバカなミスをして半日も浪費してしまったんだ!」と落胆することもあるでしょう。みんなそのようなプロセスを経て学習をしていくのです。試行錯誤した分だけ報われるので頑張ってください。

3-7 オブジェクト

今日使われる言語のほとんどはオブジェクト指向という概念をサポートしています。プログラミング学習を進めるうえで、オブジェクトに慣れることは大きな壁のひとつです。最初はとっつきにくいかもしれませんが、何度も繰り返して慣れるよう頑張ってください。

3-7-1 | オブジェクトとは

辞書で「オブジェクト」を調べると「物、物体、目的」などと説明されています。実は身の回りの「物」はすべてオブジェクトなのです。物に共通する特徴を捉え、それをプログラムで表現することで問題の解決を図ろうとするのがオブジェクト指向プログラミングの考え方です。

ピンとこないですよね？「オブジェクト」は、数字や文字のようにわかりやすい概念ではないので、最初はとっつき難いと思います。ただ、プログラミングをするうえでオブジェクトを避けて通ることはできません。以降の説明を何度も読み返し、慣れるように頑張ってください!

▶プロパティ、メソッド、インタフェース

オブジェクト指向プログラミングの学習を始める前に、特に重要な概念について最初に説明しておきます。

- オブジェクト

  「もの」です。あなたが見るもの、触るもの、すべてがオブジェクトです。ゲームにおける自分、敵、仲間、弾丸…これらもオブジェクトと考えることができます。

- プロパティ

  家の中を見回してみましょう。掃除機、炊飯器、テレビ、照明、コタツ、ドライヤーいろいろな家電製品がありますが、これらも当然オブジェクトです。では、なぜ掃除機と炊飯器を区別できるのでしょうか？ それは掃除機には掃除機の、炊飯器には炊飯器の特徴があるからです。色、大きさ、重さ、機能といったオブジェクトの個々の特徴を「プロパティ」と呼びます。

- メソッド

  オブジェクトは何らかの方法で操作できます。テレビなら電源を入れて、チャンネルを変えて、音量を調節して、という具合です。車なら、エンジンをかけて、アクセルを踏んで、ブレーキを踏んで、と操作するでしょう。このようにオブジェクトに対して働きかける個々の操作を「メソッド」と呼びます。

- インタフェース

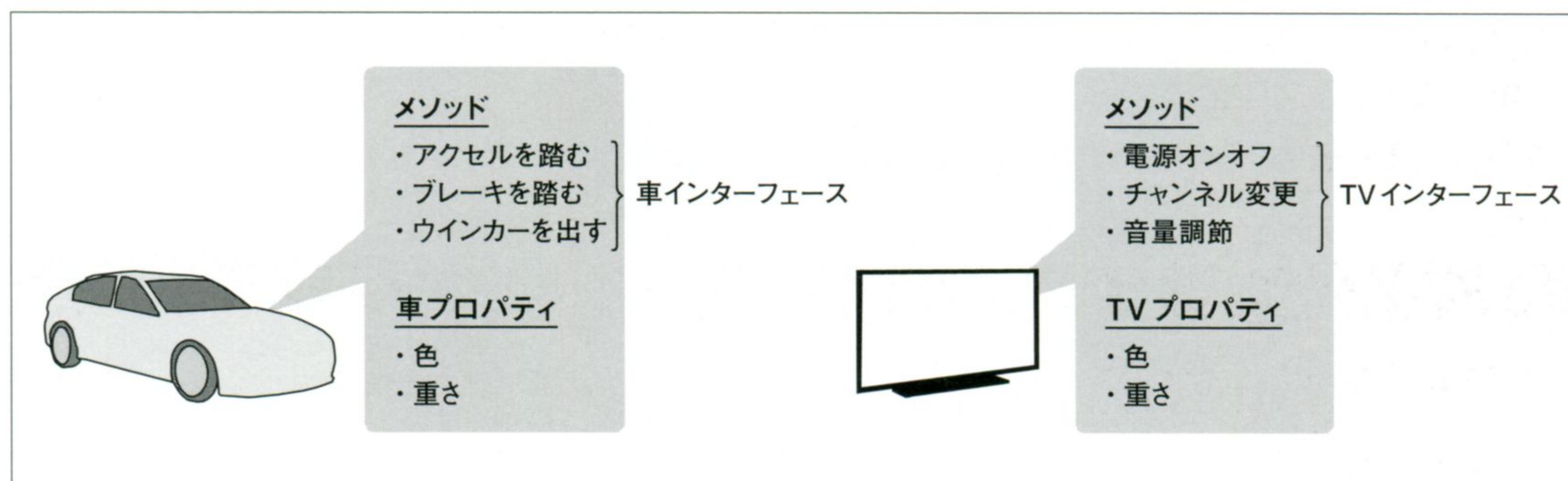
  新しくテレビを買ったとします。どのメーカーのテレビでも説明書を読まなくても使えるのは、テレビの使い方を知っているからです。新しい車を買ったとしましょう。どのメーカーの車でも直ぐに運転できるのは、車の運転方法を知っているからです。テレビの操作、車の運転といった操作には共通点があります。アク

セルとブレーキの位置が車によって違うということはありません。この意味のある操作方法のまとまりのことを「インタフェース」と呼びます。テレビのインタフェースには、電源のオン／オフ、音量のアップ／ダウンという操作があり、車のインタフェースにはエンジンのオン／オフ、ブレーキ、アクセル、ハンドル操作といったものがあります。前述したように個々の操作のことを「メソッド」と呼びますが、意味のあるメソッドの集合が「インタフェース」となります。

- **カプセル化**

  私たちがテレビのチャンネルを変えると、テレビの中では、受信周波数を変える、パケットを取り出す、画像をデコードする、というように、非常に複雑な処理が行われます。車も、アクセルを踏むと、ガソリンがエンジンに送られ燃焼系でピストンが動いてと、こちらも大変な作業が行われます。しかし、ユーザーの立場からすると、このようにテレビや車の中で起きていることを意識する必要はありません。このように中で起きていることを隠すことを「カプセル化」と呼びます。

ここまでの説明をまとめてみると以下のような図になります。

プロパティ、メソッド、インタフェース

例が日常的過ぎて、プログラムの話とはうまくむすびつかなかったかもしれません。しかし、これらの概念はオブジェクト指向プログラミングで非常に大切です。

JavaやC#といった言語では言語仕様としてインタフェースがサポートされています。一方、JavaScriptにはインタフェースという機能は用意されていません。プログラミング学習を進めるうえで把握してほしい概念だったのでここで説明しました。ご了承ください。

演習　身の回りのものから、オブジェクト、プロパティ、メソッドを考えてみよう

身の回りから適当にオブジェクトを選び、そのオブジェクトにはどのようなプロパティがあり、どんなメソッドがあるか考えてみましょう。また、そのオブジェクトのインタフェースはどのようなものでしょうか?

では、実際のプログラムで、オブジェクト指向的な考え方がどのように利用されるのかを見てみましょう。

3-7-2 | JavaScriptでのオブジェクトの定義方法

JavaScriptでのオブジェクトの定義は簡単です。単に「{」「}」の間にプロパティを宣言するだけです。

▶オブジェクトの定義

JavaScriptでは波括弧「{ }」でオブジェクトを定義します。たとえば空のオブジェクトは以下のように宣言します。

```
var empty = {};
```

空のオブジェクトではおもしろくありませんね。もう少し具体的なものをオブジェクトで表現してみましょう。長さ5cmの赤の色鉛筆があったとします。プロパティとしては、色と長さが考えられます。JavaScriptでは以下のように定義します。

```
var pen = {
    color: "red",
    length: 5
}
```

文法的には以下のように、波括弧の中に「プロパティ名:プロパティ値」と記述することでプロパティを宣言します。複数ある場合はカンマで区切ります。

オブジェクトの定義

```
オブジェクト = {プロパティ名1:プロパティ値1,プロパティ名2:プロパティ値2,…}
```

白色で時速30kmの車を定義してみましょう。その一例を以下に示します。燃料がない自動車はないのでfuelプロパティも追加してみました。

```
var car = {
    color: "white",
    speed: 30,
    fuel: 100
}
```

演習 **オブジェクトをJavaScriptで表現してみよう**

直前の演習で考えたオブジェクトをJavaScriptで表現してみましょう。

オブジェクトのプロパティにアクセスするには、オブジェクトの後ろに「.」ドットを付け、その後ろにプロパティ名を記述します。

オブジェクトのプロパティにアクセス

```
オブジェクト.プロパティ名
```

penオブジェクトのlengthプロパティを参照するサンプルを以下に示します。

SAMPLE object-pen1.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
        var pen = {
            color: "red",
            length: 5
        }
        function checkLength() {
            document.getElementById("length").textContent = pen.length;
        }
        </script>
    </head>
    <body>
        <button onclick="checkLength()">長さチェック</button>
        <p>鉛筆の長さは<span id="length"></span>cm</p>
    </body>
</html>
```

逆に、penオブジェクトのlengthプロパティに値を代入するには以下のようにします。

```
pen.length = 3;
```

オブジェクトのプロパティというと難しく聞こえるかもしれませんが、単なる変数と同じように扱うことができます。

演習 **carオブジェクトのプロパティをページに表示してみよう**

carオブジェクトのスピードとガソリンの量を表示するページを作ってください。

SAMPLE object-car1.html

▶メソッドの定義

色鉛筆は書くことができます。自動車は加速・減速・運転することができます。それらをメソッドとして定義してみましょう。先ずは色鉛筆です。「書く」という操作をdrawメソッドとして定義してみましょう。色鉛筆は描くと短くなるところがポイントです。

```
var pen = {
    color: "red",
    length: 5,
    draw: function () {
        this.length -= 0.01;
    }
}
```

メソッドの宣言方法はcolorやlengthといったプロパティ値と同じです。ただし、プロパティ値に関数を指定する点が異なります。実はJavaScriptのオブジェクトではメソッドもプロパティも同じように扱います。メソッドというと難しそうに聞こえますが、値が関数のプロパティをメソッドと呼んでいるだけなのです。

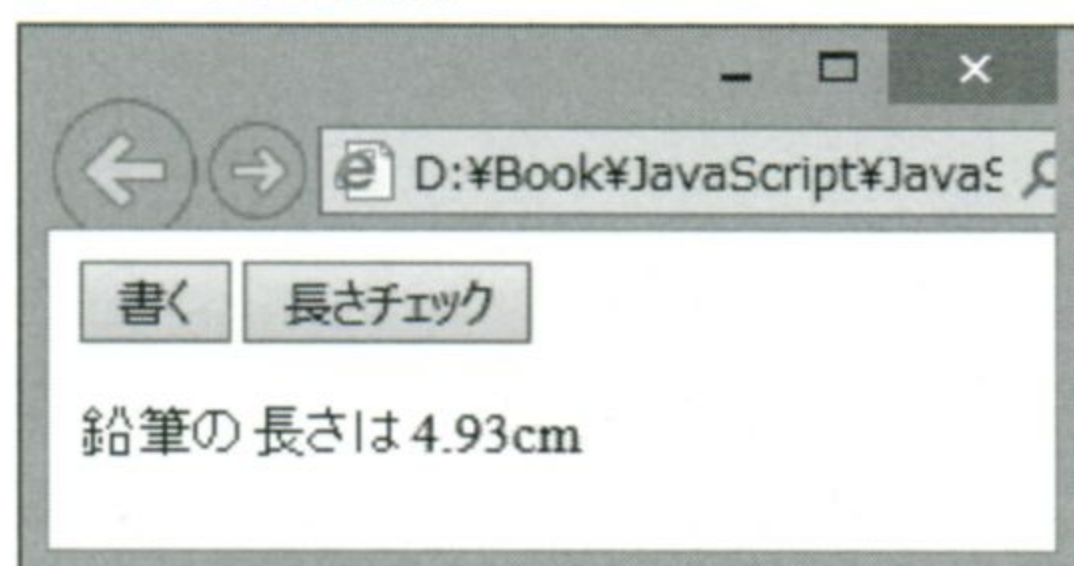
ところで、drawメソッドの中に「this」というキーワードがあります。このthisはオブジェクト自身を示すとても重要なキーワードです。世の中にはたくさんの色鉛筆がありますが、このthisは今自分が手に持っている色鉛筆を示しています。オブジェクトのプロパティは「オブジェクト.プロパティ名」で参照できると説明しました。よって、this.lengthは「自分が手に持っている色鉛筆の長さ」を参照することになります。drawメソッドでは書く度に長さを0.01cm短くしています。ほかの鉛筆の長さを短くしているのではなく、自分自身の長さを短くしていることに注意してください。

ちなみに、0.01を繰り返し引いていくとまるめ誤差が発生して4.8800000003cmのような値になることがあります。これは2進数で正確に小数を表現できないことがあるためです。

前の例にdrawメソッドを追加した例を以下に示します。

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
        var pen = {
            color: "red",
            length: 5,
            draw: function () {
                this.length -= 0.01;
            }
        }
        function stroke() {
            pen.draw();
            document.getElementById("length").textContent = pen.length;
        }
        </script>
    </head>
    <body>
        <button onclick="stroke()">書く</button>
        <p>鉛筆の長さは<span id="length"></span>cm</p>
    </body>
</html>
```

ブラウザ表示例

D:¥Book¥JavaScript¥JavaS
書く　長さチェック
鉛筆の長さは4.93cm

「書く」ボタンが押されると「pen.draw()」という行が実行されます。オブジェクトのプロパティ参照は「オブジェクト.プロパティ名」でした。オブジェクトのメソッド呼び出しは「オブジェクト.メソッド名()」となります。プロパティの値が関数なので、カッコをつけることでその関数を実行しているのです。drawメソッドの中では、自分の長さをthis.lengthで参照し、その長さを0.01cm短くしています。

ちなみに、メソッドには引数を渡すこともできます。たとえば、描画する長さを引数でわたし、それに合わせて鉛筆の長さを短くする場合は、

```
draw: function (d) {
    this.length -= 0.01 * d;
}
```

のように、メソッドの関数に引数を宣言し、その引数をメソッドの中で使用します。
呼び出し側は「pen.draw(5);」のようにメソッド呼び出し時に引数を渡します。通常の関数呼び出しと同じなのですぐに慣れるでしょう。

演習 **carオブジェクトにメソッドを定義してみよう**

carオブジェクトに加速（accelerate）、減速（decelerate）、ドライブ（drive）といったメソッドを定義し、それらの操作を可能にすると同時に、スピードとガソリンの量を表示するページを作ってください。driveメソッドは運転する距離を引数としてとるものとします。

SAMPLE object-car2.html

▶コンストラクタ

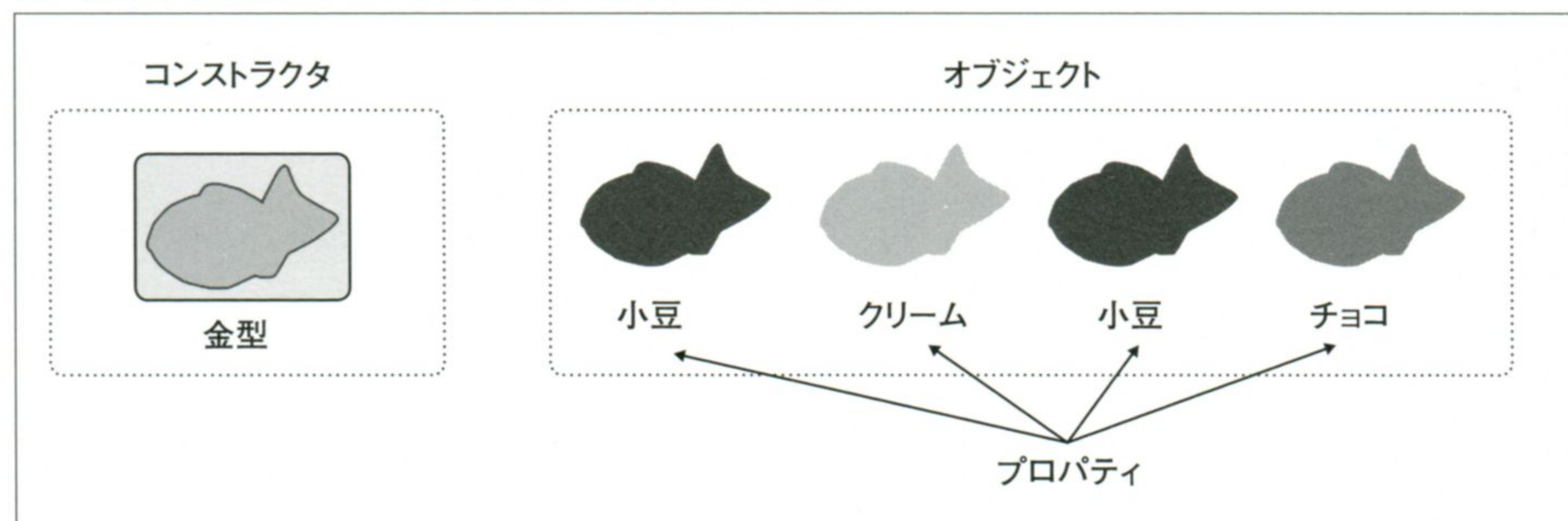
赤色のペンに加えて、青色、緑色といったペンが欲しくなりました。以下のように個々のペンについてオブジェクトを宣言することも可能です。

```
var penR = {
    color: "red",
    length: 5,
    draw: function () {
        this.length -= 0.01;
    }
}
var penG = {
    color: "green",
    length: 15,
    draw: function () {
        this.length -= 0.01;
    }
}
var penB = {
    color: "blue",
    length: 7,
    draw: function () {
        this.length -= 0.01;
    }
}
```

面倒ですよね。ペンが100本になったら対応できません。同じ種類のオブジェクトを簡単に生成するための方法が用意されています。以下のように記述することができます。

```
function Pen(color, length) {
    this.color = color;
    this.length = length;
    this.draw = function () {
        this.length -= 0.01;
    }
}
var penR = new Pen("red", 5);
var penG = new Pen("green", 15);
var penB = new Pen("blue", 7);
```

関数Penは鉛筆オブジェクトを生成するためのものです。このように特定のオブジェクトを作るために定義された関数を「コンストラクタ」と呼びます。ちょうどたい焼きの金型とたい焼きの関係に似ています。コンストラクタは金型にあたります。これをもとにたくさんのたい焼きが製造されます。たい焼きがオブジェクトに相当します。小豆、クリーム、チョコといった中身のあんはプロパティに相当します。

たい焼きの金型はコンストラクタ、たい焼きはオブジェクト

たい焼きを大量に生産する場合、金型は必須です。オブジェクトも大量に作る場合はコンストラクタを用意するのが一般的です。

上記の関数Penは普通の関数のようにも見えます。では、普通の関数とコンストラクタはどうやって区別したらよいのでしょうか? 残念ながら関数の宣言を見ただけは厳密に区別することはできません。区別は呼び出し側で行われます。関数を呼び出す際に、newというキーワードを付けるとコンストラクタとして、newがない場合は通常の関数として実行されます。上の例で、

```
var penR = new Pen("red", 5);
```

とnewをつけて呼び出しているので、関数Penはコンストラクタだということがわかるのです。コンストラクタは以下のように引数をとることができます。鯛焼きオブジェクトならあんの種類が引数になるでしょうか。個々のオブジェクトのプロパティを初期化する値を指定するのが一般的です。

```
function Pen(color, length) {
    this.color = color;
    this.length = length;
```

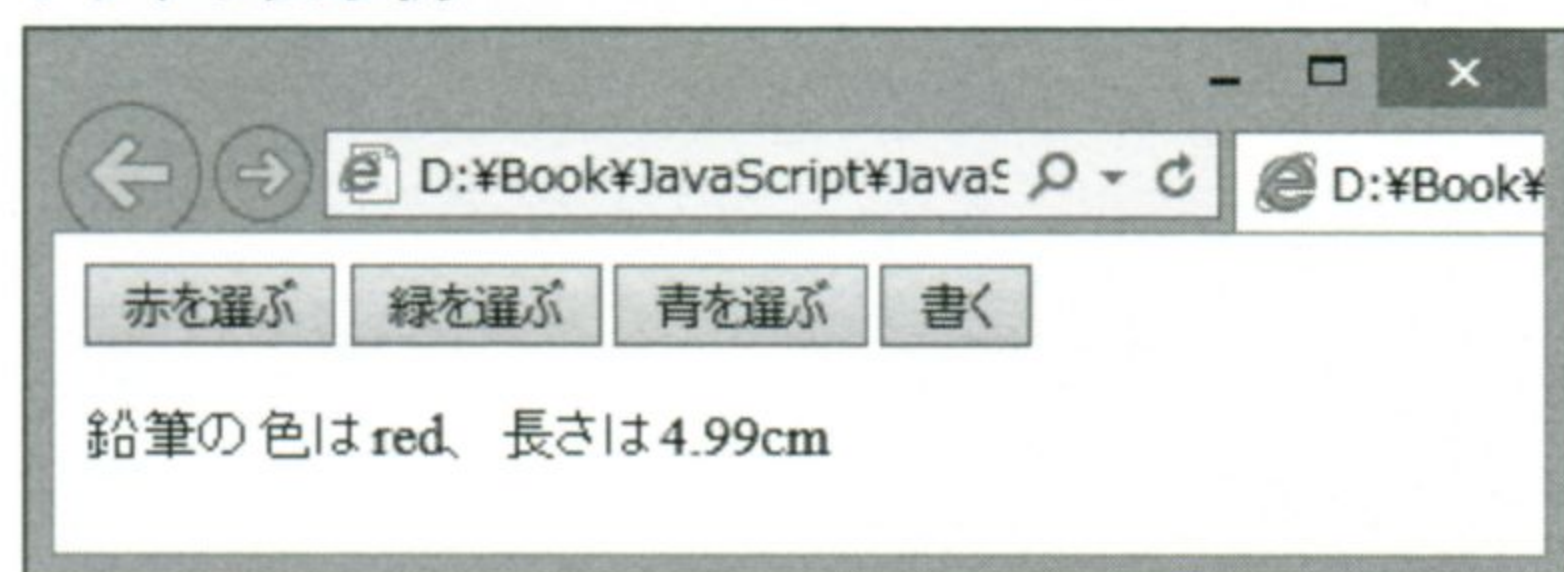
また、コンストラクタの中にthisがありますが、これは先ほど説明したように自分自身のオブジェクトを参照します。この例では、引数として受け取ったcolorを自分自身のプロパティcolorに設定するという動きになります。このthisを付け忘れるミスが非常に多いので注意してください。

コンストラクタを使ってオブジェクトを作成する例を以下に示します。

SAMPLE object-pen3.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
        function Pen(color, length) {
            this.color = color;
            this.length = length;
            this.draw = function () {
                this.length -= 0.01;
            }
        }
        var penR = new Pen("red", 5);
        var penG = new Pen("green", 15);
        var penB = new Pen("blue", 7);
        var pen = penR;

        function stroke() {
            pen.draw();
            document.getElementById("color").textContent = pen.color;
            document.getElementById("length").textContent = pen.length;
        }

        function pickR() { pen = penR; }
        function pickG() { pen = penG; }
        function pickB() { pen = penB; }
        </script>
    </head>
    <body>
        <button onclick="pickR()">赤を選ぶ</button>
        <button onclick="pickG()">緑を選ぶ</button>
        <button onclick="pickB()">青を選ぶ</button>
        <button onclick="stroke()">書く</button>
        <p>
            鉛筆の色は<span id="color"></span>、
            長さは<span id="length"></span>cm
        </p>
    </body>
</html>
```

ブラウザ表示例

D:¥Book¥JavaScript¥JavaS
赤を選ぶ 緑を選ぶ 青を選ぶ 書く
鉛筆の色はred、長さは4.99cm

演習 **実際に入力して確認してみよう**

上記コードを入力してみましょう。

3-7-3 | JavaScriptからHTMLを操作する

オブジェクトについてだいぶ慣れてきたと思います。そろそろ、JavaScriptとHTMLの連携について触れていきましょう。

▶document.getElementById()の正体

これまで、おまじないのように

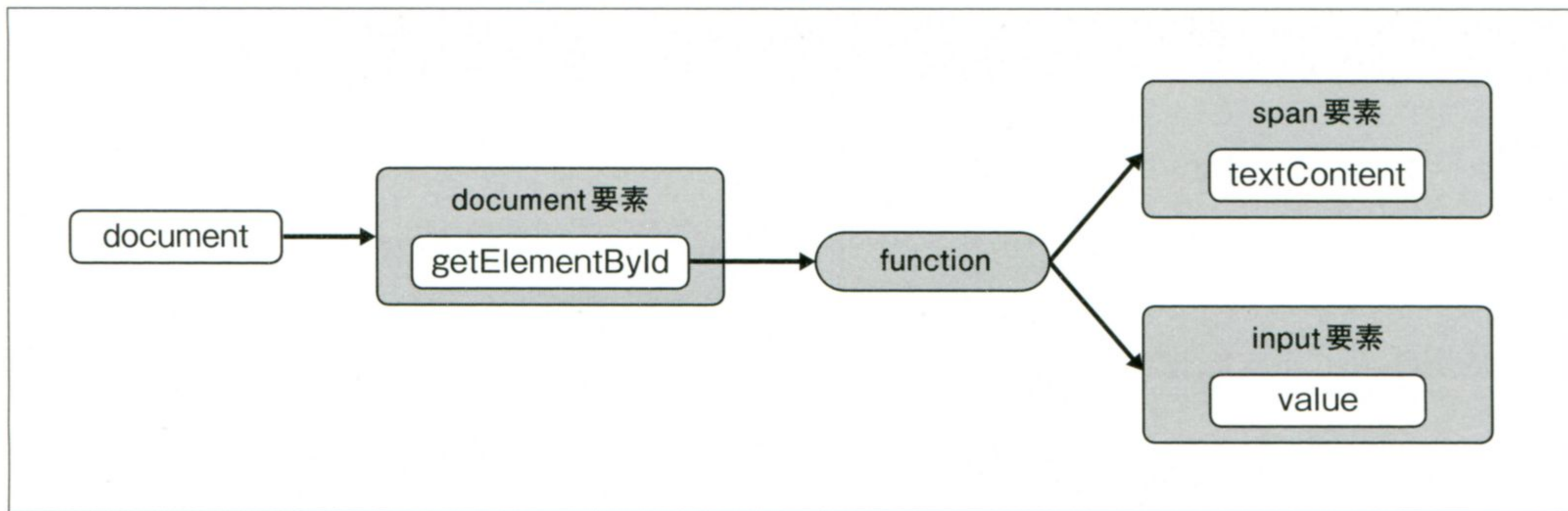
```
document.getElementById( 表示する要素のid ).textContent = 表示する値;
```

とすることで画面の表示を更新し、

```
var 取得する値 = document.getElementById( 入力要素のid ).value
```

で入力値を取得すると説明してきました。

オブジェクト、プロパティ、メソッドについて説明が終わったので、このおまじないの内容を理解する準備が整いました。HTML文書をブラウザで実行すると、ブラウザはHTML文書全体を示すオブジェクトを作成し、そのオブジェクトへの参照を広域変数documentに格納します。これはHTMLブラウザの約束事であり、どのブラウザでも同じ動きとなります。documentオブジェクトにはさまざまなプロパティ・メソッドが用意されています。getElementByIdメソッドはそのひとつで、引数で指定されたid属性をもつ要素を返します。

getElementByIdはdocumentオブジェクトのメソッドで要素を返す

getElementByIdメソッドから返される値は引数によって異なります。span要素のidを指定したらspan要素が、input要素のidを指定したらinput要素が返されます。span要素の場合は画面に表示する内容はtextContentプロパティで、input要素の場合は入力値はvalueプロパティで参照できます。

ここまでの説明を整理しましょう。

① **document** − documentオブジェクトへの参照を保持する広域変数
② **document.getElementById(id)** − documentオブジェクトのメソッド呼び出し。引数に渡されたidの要素オブジェクトを返す
③ **document.getElementById(id).textContent=表示する値** — document.getElementById(id)が返したオブジェクトのプロパティに値を代入することで画面の表示を更新

これまで1行で記述していましたが、2行で記述したほうがわかりやすいかもしれません。どちらも同じことなので、好きな方法を使ってください。

```
var obj = document.getElementById( 表示する要素のid );
obj.textContent = pen.length;
```

返された要素によって、使えるプロパティが異なる点に注意してください。たとえば、audio要素にはplayメソッドがありますが、span要素にはplayメソッドはありません。これは要素の働きを考えると納得できると思います。

では、どの要素にどのようなプロパティ、メソッドがあるのか知りたくなりますよね。残念ながらプロパティやメソッドの数は非常に多いので全部覚えるのは現実的ではありません。デバッガーを使ってください。ブレークポイントを設定し、実行を一時停止した状態で、ウオッチウインドウ（呼び方はブラウザによって異なります）に「document.getElementById("id値")」のように入力することで、その要素のプロパティやメソッドをみることができます。プロパティ名とその値、メソッドの名前から、自分が必要な情報がどこにあるか予想して試してみてください。最初は試行錯誤するかもしれませんが、そのプロセスこそが大切なのです。

Internet Explorerのデバッガでプロパティやメソッドを調べる

```
18            document.getElementById("gas
19        }
20
21        function faster() {
22            car.accelerate();
23        }
24        function slower() {
25            car.decelerate();
26        }
27        function drive() {
28            var km = document.getElement
29            car.drive(km);
30        }
31        </script>
32    </head>
33    <body>
34        <button onclick="faster()">加速<
35        <button onclick="slower()">減速<
36        <br />
```

ウォッチ	
document	[object HTMLDocument]
document.getElementById("distance")	[object HTMLInputElement]
[Methods]	
accept	""
accessKey	""
align	""
alt	""
ATTRIBUTE_NODE	2
attributes	[object NamedNodeMap]
autocomplete	""
autofocus	false
border	""
canHaveChildren	true

演習

オブジェクトのメソッド、プロパティを調べてみよう

documentオブジェクトにはどんなメソッド・プロパティがあるか見てみましょう。また、HTMLの中の適当な要素を参照し、どんなメソッド・プロパティがあるか見てみましょう。

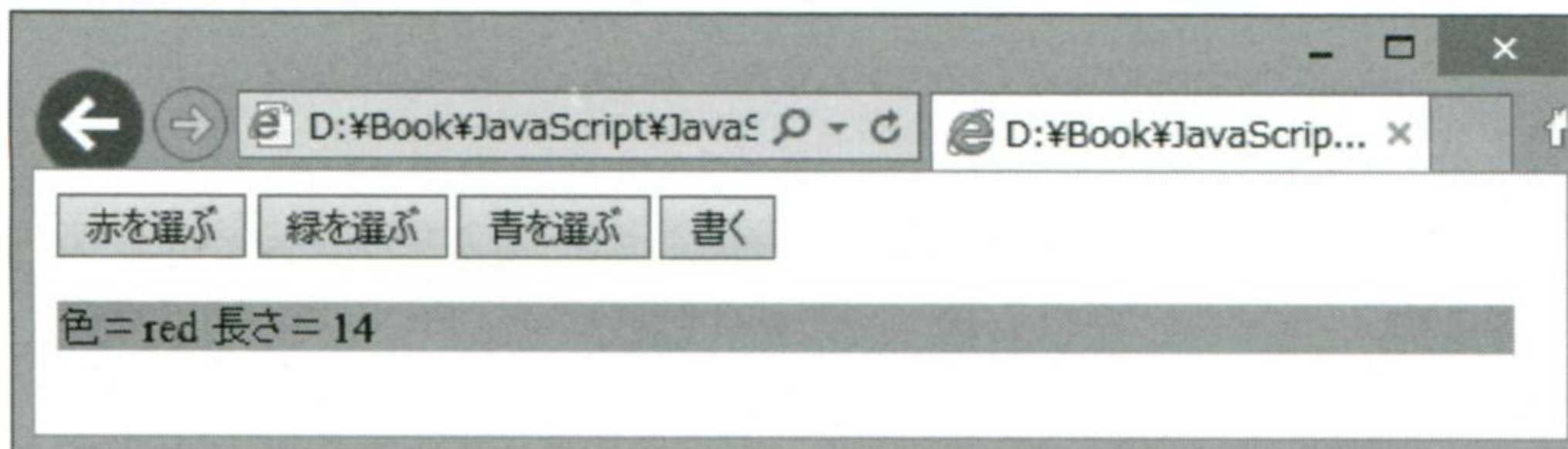
3-7-4 | JavaScriptからCSSを操作する

これまで、JavaScriptを利用して文字を表示したり、値を取得したりする方法を説明してきました。次にCSSのプロパティをJavaScriptから操作する方法について説明しましょう。

SAMPLE object-pen4.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
        function Pen(color, length) {
            this.color = color;
            this.length = length;
            this.draw = function () {
                this.length -= 1;
            }
        }
        var penR = new Pen("red", 15);
        var penG = new Pen("green", 20);
        var penB = new Pen("blue", 8);
        var pen = penR;
```

```
    function stroke() {
        pen.draw();
        var pencil = document.getElementById("pencil");
        pencil.style.width = pen.length + "cm";                    ←1
        pencil.style.backgroundColor = pen.color;
        pencil.textContent = "色＝" + pen.color + " 長さ＝" + pen.length;
    }

    function pickR() { pen = penR; }
    function pickG() { pen = penG; }
    function pickB() { pen = penB; }
    </script>
  </head>
  <body>
    <button onclick="pickR()">赤を選ぶ</button>
    <button onclick="pickG()">緑を選ぶ</button>
    <button onclick="pickB()">青を選ぶ</button>
    <button onclick="stroke()">書く</button>

    <p id="pencil" style="background-color:red; width:5cm;"></p>      ←2
  </body>
</html>
```

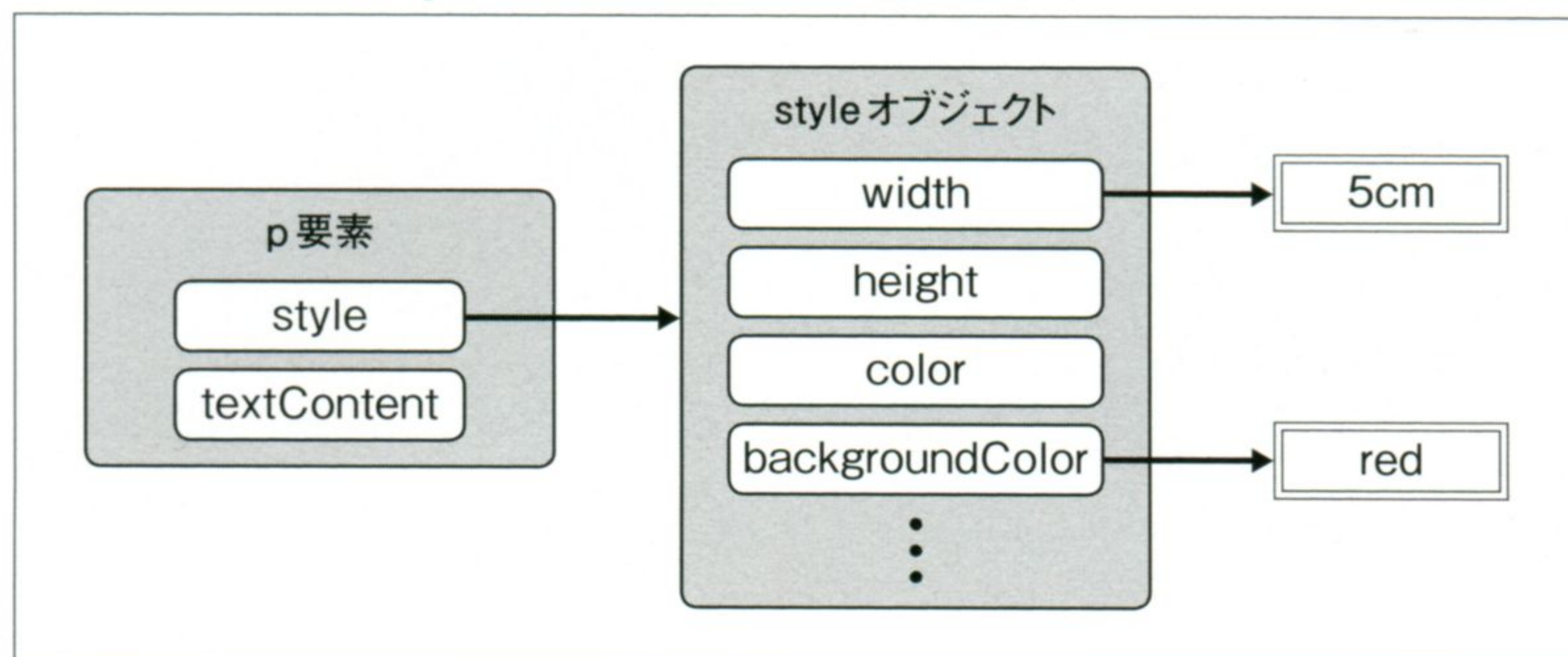
ブラウザ表示例

「書く」ボタンを押すと、色と長さが変化します。これはJavaScriptからCSSの値を操作しているからです。その動作手順について見ていきましょう。まず、HTMLでは2のように色と長さを初期化しています。

```
<p id="pencil" style="background-color:red; width:5cm;"></p>
```

次に、1のstroke関数を見てください。

```
function stroke() {
    pen.draw();
    var pencil = document.getElementById("pencil");        ←3
    pencil.style.width = pen.length + "cm";
    pencil.style.backgroundColor = pen.color;
    pencil.textContent = "色=" + pen.color + " 長さ=" + pen.length;
}
```

3で「id="pencil"」という要素、すなわち<p>要素のオブジェクトを取得しています。要素のCSS特性はstyleプロパティで参照されるオブジェクトに格納されています。そのオブジェクトの中の個々のCSS特性の値を参照すれば現在のCSS特性の値が取得でき、値を代入するとCSS値が更新されます。その様子を以下の図に示します。

要素のプロパティはstyleオブジェクトでまとめて管理されている

ここで一点注意してほしいのは、CSS特性の名前と、JavaScriptでのプロパティ名の対応関係です。widthやheightといった1単語からなる名前の場合は、どちらも同じ名前を使用します。しかしながら、「CSS特性名に「-」（ハイフン）が含まれる場合、JavaScriptではハイフンを削除して、その次の文字を大文字にする」というルールがあります。例を見てみるのが一番わかりやすいでしょう。

これは、JavaScriptではハイフンはマイナス演算子として予約されているためです。

CSS特性の名前とJavaScriptでのプロパティ名

CSS特性名	JavaScriptのプロパティ名
background-color	backgroundColor
border-color	borderColor
margin-left	marginLeft

3-7-5 | DOM (Document Object Model)

最後に、JavaScriptから要素をつくったり、削除したりする方法について見ていきましょう。ちなみに、DOMとはDocument Object Modelの略で、HTML文書をオブジェクトとして操作する方法を定めたものです。難しく聞こえますが、単にJavaScriptからHTML文書を更新する方法だと思ってください。

HTMLやCSSの章で、HTML文書は<html>要素をトップとする階層構造をとるという説明をしました。これまで、document.getElementById()メソッドを使い、文書中に宣言されている要素への参照を取得してきました。実は、documentオブジェクトにはgetElementById()以外にもさまざまなメソッドが用意されています。

▶要素を生成するcreateElement()、子要素を挿入するappendChild(child)

document.createElement()メソッドを使うと、要素をJavaScriptから生成することができます。例を見てみましょう。

SAMPLE object-create1.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            var colors = ["red", "blue", "green", "yellow", "purple"];
            var index = 0;
            function insert() {
                var parent = document.getElementById("mylist");     ←1
                var item = document.createElement("li");     ←2
                item.textContent = colors[index];
                item.style.color = colors[index];
                index = ++index % colors.length;
                parent.appendChild(item);     ←3
            }
        </script>
    </head>
    <body>
        <button onclick="insert()">挿入</button>
        <ol id="mylist">     ←4
        </ol>
    </body>
</html>
```

insert()関数の中で起きていることに着目してください。変数parent 1 はmylistというidを持つ要素 4 、すなわち<ol>への参照を保持します。

次の 2 「var item = document.createElement("li")」で、<li>要素を作成し、その参照を変数itemに代

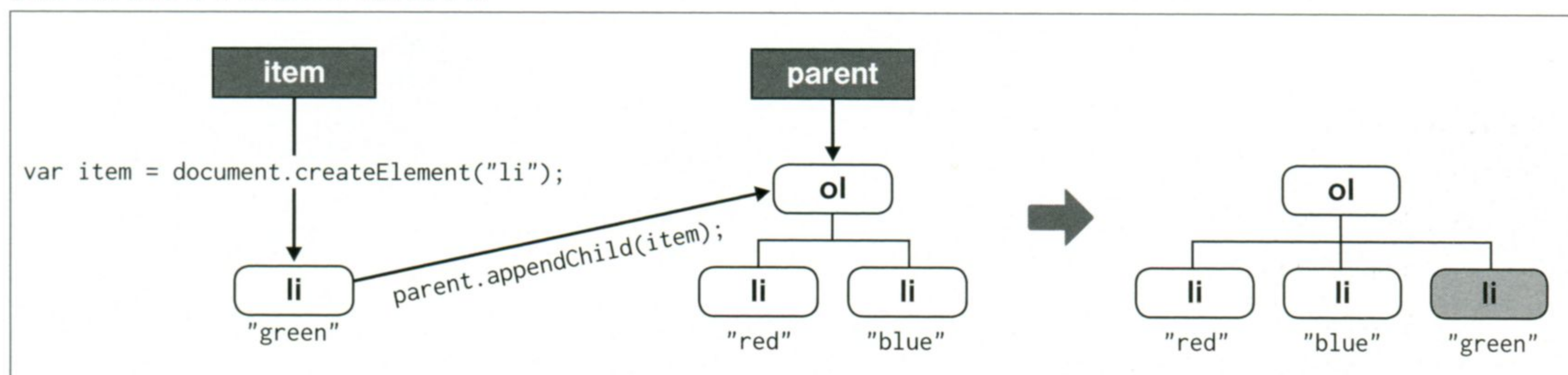
入しています。これで、JavaScriptからHTML要素が作成できました。

しかし、この段階ではまだ画面上に<li>要素は表示されません。なぜなら、画面上のどこに表示してよいかわからないからです。HTMLは階層構造だったことを思い出してください。今回作成した<li>要素も、階層構造のどこかに挿入しなくてはなりません。その処理を実行しているのが3の文です。

```
parent.appendChild(item)
```

appendChildとは子供を追加するという意味です。parentは<ol>要素への参照を保持していました。itemは新規に作成した<li>要素です。つまり、<ol>要素に対して、<li>を子供として追加せよと命令しているのです。この行を実行することで、新しく作成した<li>要素が、<ol>要素に挿入され、画面が更新されることになります。その様子を以下の図に示します。

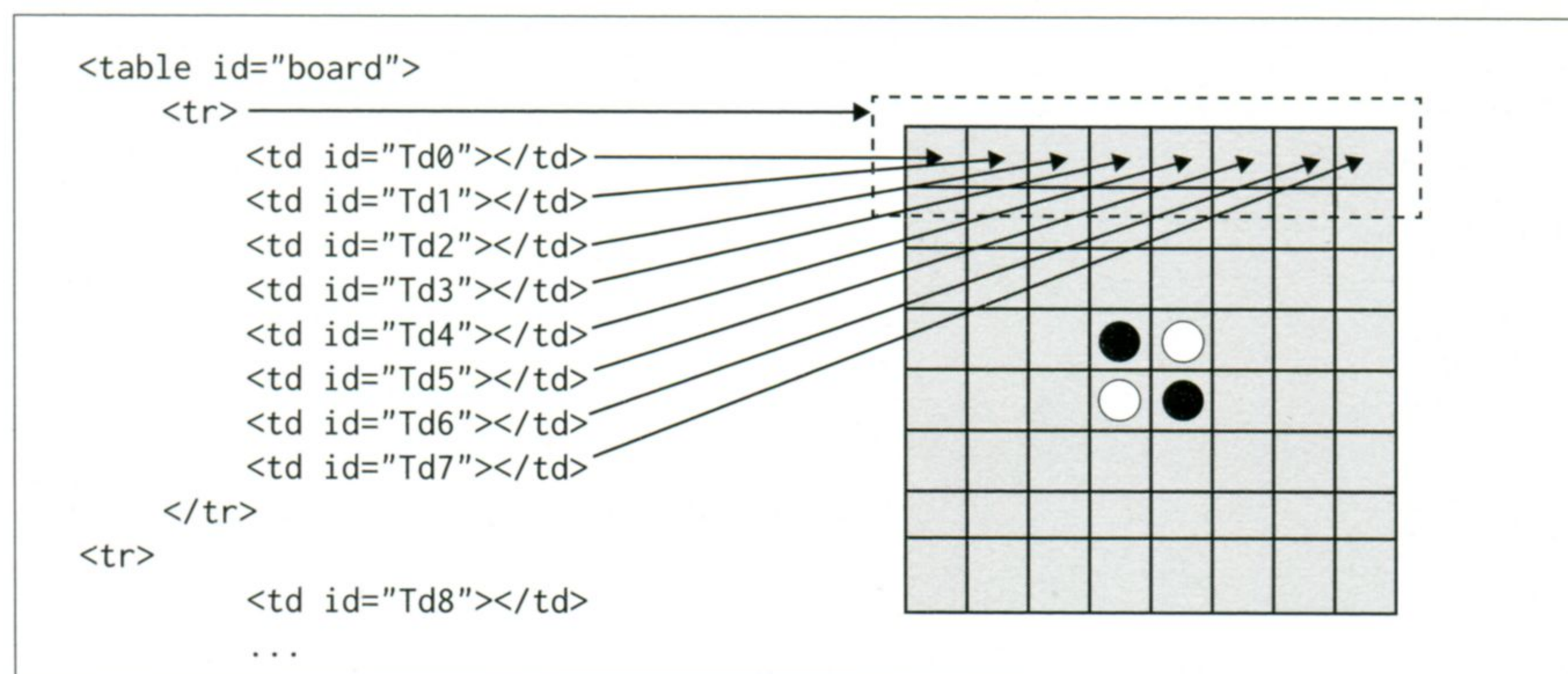
要素を作成して子要素として挿入する

item
var item = document.createElement("li");
li
"green"
parent.appendChild(item);
parent
ol
li li
"red" "blue"
ol
li li li
"red" "blue" "green"

appendChildは文字どおり「追加する」という意味なので、最後の子要素として追加されます。

演習　プログラムを入力して確認してみよう

サンプルのプログラムを入力して要素が追加される様子を確認してみましょう。

同じような要素を数多く記述するような場合に、この手法が威力を発揮します。たとえば、8×8のゲーム盤を考えてみましょう。HTMLのTable要素を使って記述すると以下のようになります。

8×8のゲーム盤

```
<table id="board">
    <tr>
        <td id="Td0"></td>
        <td id="Td1"></td>
        <td id="Td2"></td>
        <td id="Td3"></td>
        <td id="Td4"></td>
        <td id="Td5"></td>
        <td id="Td6"></td>
        <td id="Td7"></td>
    </tr>
<tr>
        <td id="Td8"></td>
        ...
```

8個の<tr>要素、それぞれについて8個の<td>要素、合計64個の<td>要素を繰り返し記述しなくてはなりません。こういった単純な繰り返し作業こそコンピュータに任せるべきです。

SAMPLE ReversiMock.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <style>
        #board {
            background-color:black;
        }
        td.cell {
            background-color: green;
            width: 30px;
            height: 30px;
        }
    </style>
    <script>
        function init() {
            var b = document.getElementById("board");
            for (var i = 0 ; i < 8 ; i++) {
                var tr = document.createElement("tr");          ←1
                for (var j = 0 ; j < 8 ; j++) {
                    var td = document.createElement("td");      ←2
                    td.className = "cell";
                    td.id = "cell" + i + j;                       ←4    ←5
                    td.onclick = clicked;
                    tr.appendChild(td);  ←3
                }
                b.appendChild(tr);       ←6
            }
        }
        function clicked(e) {
            document.getElementById("info").textContent = e.target.id + " clicked";
        }
    </script>
</head>
<body onload="init()">
    <table id="board"></table>
    <h2 id="info"></h2>
</body>
</html>
```

1「var tr = document.createElement("tr")」で<tr>要素を作り、変数trに格納しています。

その後、2の「var td = document.createElement("td")」で要素を作り、3の「tr.appendChild(td)」でtrの子要素に挿入しています。これを内側のfor文4で8回繰り返すことで1行の要素を作っています。

さらに、外側のfor文5で8行分の要素をつくり、それぞれの行を6の「b.appendChild(tr)」で<table>に挿入しています。盤面を作るためのコード量を大幅に削減できました。

なお、HTMLで、ID属性、CSSクラス属性、クリック時のイベントハンドラを設定する場合は、

```
<td id="cell15" class="cell" onclick="clicked()"></td>
```

のように記述しましたが、JavaScriptでは

```
var td = document.createElement("td");
td.className = "cell";
td.id = "cell15";
td.onclick = clicked;
```

のように記述します。クリック時には要素のonclickプロパティに登録した関数が呼び出されます。この関数をイベントハンドラと呼びます。イベントハンドラの中では、クリックされた要素への参照を e.target で取得しています。イベントハンドラは後ほど、関数オブジェクトでより詳しく説明しますので、ここでは「そんなものか」と思っておいて下さい。

演習　自分でゲーム盤をつくってみよう

兄弟でマルバツゲームがやりたいと駄々をこね始めました。勝敗の判定は自分たちでできるので、クリックすると○と×が表示されれば満足なようです。そんなページを作ってみてください。

SAMPLE TicTacToe.html

3-7-6 | タイマー関連のメソッド

本節の最後に、一定時間後に処理を実行したり、一定間隔で処理を実行したりする、タイマー関連の主なメソッドを紹介しましょう。

タイマー関連のメソッド

メソッド	説明
setTimeout(func, msec)	msecミリ秒後に関数funcを1回呼び出す。戻り値としてtimerIdをかえす
clearTimeout(tid1)	setTimeoutの処理を停止する。tid1はsetTimeoutの戻り値
setInterval(func, msec)	msecミリ秒間隔で関数funcを繰り返し呼び出す。戻り値としてtimerIdをかえす
clearInterval(tid2)	setIntervalの処理を停止する。tid2はsetIntervalの戻り値

実はこれらの関数はwindowオブジェクトのメソッドです。よって、厳密には「window.setTimeout(関数名, ミリ秒)」と書くのが正しい書き方なのですが、windowオブジェクトは省略することができます。ゲームでよく使うのはsetInterval()とclearInterval()です。

SAMPLE timer.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            var timerId = NaN, count = 0;
            function startTimer() {
                timerId = setInterval(tick, 1000);
            }
            function stopTimer() {
                clearInterval(timerId);
            }
            function tick() {
                count++;
                document.getElementById("counter").textContent = count;
            }
        </script>
    </head>
    <body>
        <button onclick="startTimer()">スタート</button>
        <button onclick="stopTimer()">ストップ</button>
        <h2 id="counter"></h2>
    </body>
</html>
```

ブラウザ表示例

D:¥Book¥JavaScript¥Ja
スタート ストップ
7

「スタート」ボタンを押すと

```
timerId = setInterval(tick, 1000);
```

が実行され、1秒ごとに関数tickが呼び出され、数字のカウントが始まります。ストップを押すと

```
clearInterval(timerId)
```

が実行され止まります。ちゃんと動いているように見えます。しかし、スタートボタンを連打すると、カウンタ増加のスピードが速くなり、ストップを押しても止まらなくなります。なぜだかわかりますか?

「スタート」ボタンを連打するとストップが効かなくなる

●タイマーがひとつであれば意図した正常に動作する

スタート　tick()　1秒　tick()　1秒　tick()　ストップ

●スタートボタンを連打すると、同時に複数のタイマーが動き始める

スタート　tick()　1秒　tick()　1秒　tick()　1秒　tick()

スタート　tick()　1秒　tick()　1秒　tick()

スタート　tick()　1秒　tick()　ストップ

スタートボタンが押されるとタイマーが開始されます。タイマーがひとつであれば意図した挙動になります。しかし、スタートボタンを連打すると、同時に複数のタイマーが動き始めます。ストップを押してクリアされるのは最後のタイマーだけです。このような理由でカウンタが止まらなかったのです。このような状況を回避するために、タイマーを開始するときは最初にタイマーをクリアする習慣をつけるとよいでしょう。

```
function startTimer() {
    clearInterval(timerId);
    timerId = setInterval(tick, 1000);
}
```

ほぼすべてのリアルタイムゲームではsetInterval()を呼び出し、定期的に特定の関数を実行しています。詳しくは後半のゲームを参照してください。

ところで、ときどきsetIntervalで関数を呼び出す間隔を10mscや5msecといった非常に小さな値にしているケースを見かけます。残念ながらあまりお勧めできません。多くの液晶ディスプレイの更新間隔は60ヘルツです。すなわち1秒間に60回画面が書き換えられるということになります。1/60 = 16.666msecなので、これ以上高い頻度でタイマーを更新しても無意味なことが多いからです。呼び出し間隔を短くするほどCPUに負荷がかかります。むやみに更新間隔を短くするのではなく、適当な更新間隔を見つけることが大切です。

3-8 組み込みオブジェクト

これまで自分でオブジェクトを作る方法や、HTML文書の要素をオブジェクトとして操作する方法について見てきました。実は、JavaScriptにはあらかじめいくつかのオブジェクトが用意されています。このようなオブジェクトを「組み込みオブジェクト」と呼びます。

3-8-1 Date オブジェクト

日付や時刻を扱うためのオブジェクトです。日時・時刻の取得や設定のために数多くのメソッドを提供しています。

SAMPLE date0.htm

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            function update() {
                var now = new Date();
                document.getElementById("year").textContent = now.getFullYear();
                document.getElementById("month").textContent = (now.getMonth() + 1);
                document.getElementById("date").textContent = now.getDate();
                document.getElementById("hour").textContent = now.getHours();
                document.getElementById("min").textContent = now.getMinutes();
                document.getElementById("sec").textContent = now.getSeconds();
                document.getElementById("msec").textContent = now.getMilliseconds();
            }
        </script>
    </head>
    <body onload="update()">
        <p>
            <span id="year"></span>年
            <span id="month"></span>月
            <span id="date"></span>日
            <span id="hour"></span>時
            <span id="min"></span>分
            <span id="sec"></span>秒
            <span id="msec"></span>ミリ
        </p>
```

```
        <button onclick="update()">更新</button>
    </body>
</html>
```

ブラウザ表示例

2015年 7月 13日 21時 25分 49秒 500ミリ

更新

「var now = new Date()」とすると現在時刻のDateオブジェクトが生成されます。主なメソッドに以下のようなものがあります。

Dateオブジェクトの主なメソッド

メソッド	説明
getFullYear()	西暦の年を返す
getMonth()	月を返す。1月は0、2月は1…のように0から始まることに注意
getDate()	日を返す
getHours()	時を返す
getMinutes()	分を返す
getSeconds()	秒を返す
getMilliseconds()	ミリ秒を返す
getTime()	1970年1月1日00:00:00UTCからの経過ミリ秒を返す

演習 ストップウォッチで経過時間を表示するページをつくってみよう

ストップウォッチで10秒ちょうどを計る遊びをしたことがあると思います。スタートボタン、ストップボタンを用意し、スタートボタンを押してからストップボタンを押すまでの経過時間を表示するページをつくってください。それぞれのボタンを押下したときに、別々のDateオブジェクトを作成し、それぞれのgetTime()メソッドの戻り値の差分を計算することで経過時間を求めることができます。

SAMPLE date1.htm

3-8-2 ｜ Math オブジェクト

各種計算を行うためのメソッドを提供しています。主なメソッドに以下のようなものがあります。

Mathオブジェクトの主なメソッド

メソッド	説明
Math.min(a, b)	aとbの小さいほうを返す
Math.max(a, b)	aとbの大きいほうを返す
Math.random()	0以上1未満の乱数を発生させる
Math.floor(n)	小数点以下の値を切り捨てた値を返す
Math.ceil(n)	小数点以下の値を切り上げた値を返す

ゲームに乱数は欠かせません。サイコロなら0～5まで、トランプなら0～12までのランダムな数値を生成する必要があるでしょう。Mathオブジェクトにはrandom()というメソッドがありますが、生成される値は0から1未満の小数です。これをある一定の範囲の整数にマッピングするためにはMath.floor(n)を利用します。0～5までの6つの乱数を生成する例を以下に示します。

SAMPLE random0.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
        function random() {
            var r = Math.floor(Math.random() * 6);
            document.getElementById("value").textContent = r;
        }
        </script>
    </head>
    <body>
        <button onclick="random()">乱数:<span id="value"></span></button>
    </body>
</html>
```

演習　2～5の乱数を生成してみよう

上記プログラムは0～5の乱数を生成しました。上のプログラムに1行追加して、いったん生成した乱数rが2から5の範囲に収まるようにしてください。

SAMPLE random1.html

3-8-3 Arrayオブジェクト

JavaScriptで配列を作った場合、それらは必ずArrayオブジェクトとなります。よって、Arrayオブジェクトのプロパティやメソッドがもれなく利用できます。たとえば、どんな配列でもlengthプロパティで配列のサイズが取得できますが、これは配列がArrayオブジェクトだからです。

主なメソッドを以下に列挙します。

Arrayオブジェクトの主なメソッド

メソッド	説明
push(a)	配列の最後に要素aを追加する
pop()	配列の最後の要素を削除して返す
shift()	先頭の要素を削除して返す
indexOf(検索対象)	引数の検索対象を先頭から探し、その番号を返す。ない場合は-1を返す
lastIndexOf(検索対象[, 検索位置])	引数の検索対象をうしろから探し、その番号を返す。ない場合は-1を返す
splice(index, howMany)	indexからhowMany個分の古い要素を取り除く

これらのメソッドの動きを確認するページを以下に示します。

SAMPLE array0.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            var data = [1, 8, 5, 7, 2, 6, 7, 4, 0];

            function push() {
                var v = Math.floor(Math.random() * 10);
                var r = data.push(v);
                update(r);
            }
            function pop() {
                var r = data.pop();
                update(r);
            }
            function shift() {
                var r = data.shift();
                update(r);
            }
            function splice() {
                var r = data.splice(3, 2);
```

```
                update(r);
            }
            function update(rval) {
                document.getElementById("data").textContent = data;
                document.getElementById("length").textContent = data.length;
                document.getElementById("rval").textContent = rval;
                var v0 = data.indexOf(7);
                var v1 = data.lastIndexOf(7);
                document.getElementById("i0").textContent = v0;
                document.getElementById("i1").textContent = v1;
            }
        </script>
    </head>
    <body>
        <button onclick="push()">push()</button>
        <button onclick="pop()">pop()</button>
        <button onclick="shift()">shift()</button>
        <button onclick="splice()">splice(3, 2)</button>

        <ul style="font-size:20px;">
            <li>配列データ：<span id="data"></span></li>
            <li>lengthプロパティ：<span id="length"></span></li>
            <li>戻値：<span id="rval"></span></li>
            <li>indexOf(7)：<span id="i0"></span></li>
            <li>lastIndexOf(7)：<span id="i1"></span></li>
        </ul>
    </body>
</html>
```

ブラウザ表示例

D:¥Book¥JavaScript¥JavaS

push() pop() shift() splice(3, 2)

- 配列データ:1,8,5,7,2,6,7,4,0,4
- lengthプロパティ:10
- 戻値:10
- indexOf(7):3
- lastIndexOf(7):6

演習　上記プログラムを実際に入力して実行してみよう

上記プログラムを入力して実行してください。それぞれのメソッドの働きと戻り値を確認してください。

3-8-4 Stringオブジェクト

配列がArrayオブジェクトであったように、JavaScriptでの文字列はStringオブジェクトとなります※。また、配列の要素数をlengthプロパティで取得できたのと同様に、文字列の長さもlengthプロパティで取得できます。主なメソッドを以下に列挙します。

※ 厳密には単なる文字列とオブジェクトは異なります。しかしながら、暗黙な型変換が行われるため、文字列もオブジェクトのように扱うことができます。

Stringオブジェクトの主なメソッド

メソッド	説明
charAt(i)	i番目の文字を返す
indexOf(c)	引数の検索対象cを先頭から探し、その番号を返します。ない場合は-1を返す
lastIndexOf(c)	引数の検索対象cをうしろから探し、その番号を返す。ない場合は-1を返す
startsWith(str)	引数で指定された文字列で開始されているかを返す
substr(start, length)	指定した位置からlength文字数分返す

文字列オブジェクトを使ったサンプルを以下に示します。動いている様子を実際に確認するのが一番です。

SAMPLE kaibun.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            var data = [
                { str: "タケヤブ ヤケタ", jpn: "竹藪焼けた" },
                { str: "タケムラタケコ コケタラムケタ", jpn: "竹村武子こけたら剥けた" },
                { str: "ナタデココデタナ", jpn: "ナタデココ出たな" },
                { str: "リカガカリ", jpn: "理科係" },
                { str: "イカノダンスハスンダノカイ", jpn: "烏賊のダンスは済んだのかい" },
                { str: "ヨノナカネ カオカオカネカナノヨ", jpn: "世の中ね顔かお金かなのよ" },
            ];
            var timerId, index = 0, pos = 0;

            function start() {
                pos = 0;
                document.getElementById("jpn").textContent = "";
                index = (index + 1) % data.length;
                clearInterval(timerId);
                timerId = setInterval(tick, 200);
            }
```

```
            function tick() {
                var str = data[index].str;
                document.getElementById("str").textContent = str.substr(0, pos);
                if (++pos > str.length) {
                    clearInterval(timerId);
                    document.getElementById("jpn").textContent = data[index].jpn;
                }
            }
        </script>
    </head>
    <body>
        <button onclick="start()">スタート</button>
        <h2 id="str"></h2>
        <h2 id="jpn"></h2>
    </body>
</html>
```

ブラウザ表示例

D:¥Book¥JavaScript¥JavaS

スタート

イカノダンスハスン

3-9 プロトタイプ

多くのオブジェクト指向言語では「継承」という仕組みを使って、オブジェクトを再利用しますが、JavaScriptでは「プロトタイプ」という仕組みを使います。プロトタイプはJavaScriptの学習において大きな壁のひとつです。本書では類書とはちょっと違った視点から説明してみたいと思います。

3-9-1 プロトタイプとは

JavaScriptで文字列を作ると、charAt()メソッドを使って該当する場所の文字が取得できます。

```
var str = "Hello world";     // 文字列オブジェクトを作成
var c0 = str.charAt(0);      // c0は'H'
var c1 = str.charAt(1);      // c1は'e'
```

一方、配列を作ると、push()/pop()といったメソッドを利用してデータの出し入れができます。

```
var data = [];               // 配列オブジェクトを作成
data.push(1);                // '1'を追加
var d1 = data.pop();         // d1は'1'
```

しかし、配列に対してcharAt()メソッドを呼び出すことはできません。反対に、文字列に対してpush()やpop()といったメソッドを呼び出すこともできません。なぜでしょうか?

```
var str = "Hello world";     // 文字列オブジェクトを作成
str.push(2);         // エラー
```

ほかのオブジェクト指向言語をご存じの方であれば「型が違うから呼び出せるメソッドが違うのは当たり前でしょ?」と考えるかもしれません。しかし、JavaScriptには厳密な型はないので、この答えは正しくありません。この挙動を理解する鍵がプロトタイプなのです。

JavaScriptでオブジェクトを生成すると、そのオブジェクトにはprototypeという見えないプロパティが暗黙のうちに追加され、そのプロパティは別のオブジェクトを参照します。たとえば、以下のように文字列オブジェクトを作成したとします。

```
var str0 = new String("Hello");
```

すると、このstr0オブジェクトには、暗黙のプロパティprototypeが設定され、別のオブジェクトを参照します。

暗黙のプロパティ prototype が別のオブジェクトを参照する

ここで、もうひとつ別の文字列オブジェクトを作ってみます。

```
var str1 = new String("World");
```

オブジェクトの関係は以下のようになります。str0とstr1はまったく別のオブジェクトなのに、prototypeの参照先は同一であることに着目してください。

prototypeの参照先は同一

このprototypeの参照先オブジェクトがどのようなものか気になりますよね？ しかし、

```
p0 = str0.prototype;    // p0はundefined
```

としても参照先にたどり着くことはできません。当初、JavaScriptにおいて、prototypeは「暗黙の参照先」として設計されたこともあり、プログラムで明示的に辿ることはできませんでした。とはいうものの、やはりこのprototypeにアクセスしたいという需要には強いものがありました。そのような状況を受けて、prototypeへの参照を辿るためのメソッドが正式に規定されました。以下のように記述することでprototypeへの参照が取得できます。

```
p0 = Object.getPrototypeOf(str0);
```

str0とstr1のprototypeが同じオブジェクトを参照しているか確認してみましょう。

```
var str0 = new String("Hello");
p0 = Object.getPrototypeOf(str0);

var str1 = new String("World");
p1 = Object.getPrototypeOf(str1);

var isIdentical = (p0 === p1);  // trueになります
```

「===」は厳密な比較をする演算子です。オブジェクトの比較に使用した場合、左辺と右辺が同じオブジェクトか否かを比較します。上記の例では、p0もp1も同じオブジェクトを参照しているのでisIdenticalはtrueとなります。

prototypeの参照先であるp0オブジェクトが取得できました。では、これにどのようなプロパティがあるかデバッガを使って見てみましょう。

デバッガでp0オブジェクトのプロパティを確認

p0のメソッド

画面右側にp0のメソッド一覧が表示されています。charAt, charCodeAt, concatなどStringオブジェクト固有のメソッドが定義されていることがわかります。

つまり、文字列オブジェクトを作成した場合、charAtなどのメソッドが呼べたのは、prototypeで参照されるオブジェクトにcharAtメソッドが定義されていたからなのです。

でもちょっと待ってください。前の例では、文字列オブジェクトstr0に対してcharAtメソッドを呼び出しました。prototypeオブジェクトにcharAtメソッドがあることは確認できましたが、str0オブジェクトにcharAtメソッドがないことに変わりはありません。

charAtメソッドはprototypeオブジェクトにはあるがstr0オブジェクトにはない

charAt
str0
"Hello"
prototype
charAt関数
charAt関数

(3-9-2 | プロトタイプ継承)

なぜstr0.charAt(0)のように、str0に対してcharAt()メソッドが呼び出せたのでしょうか? これこそが、JavaScriptのプロトタイプ継承にほかなりません。

JavaScriptではプロパティが参照されたとき、

- プロパティがそのオブジェクトで定義されていれば、そのプロパティを参照する
- 定義されていなければprototypeで参照される先のオブジェクトで同じことを行う

という動作をprototypeが辿れなくなるまで繰り返します。

今回の場合、str0オブジェクトのcharAt()メソッドが参照されました。しかし、str0オブジェクトにはそのようなメソッドが定義されていなかったため、prototypeの参照先を見て、そこにあったcharAt()が呼び出されたのです。

別の例でさらに詳しく見てみましょう。ObjectAに対してstart()、sayHello()、startPlay()といったメソッドを呼び出したとします。

ObjectAに対してstart()、sayHello()、startPlay()を呼び出す

start
sayHello
startPlay
ObjectA
prototype
ObjectB
prototype
ObjectC
start関数
sayHello関数
startPlay関数

start()から始めましょう。ObjectAにはstart()メソッドがあるので、普通に呼び出すことができます。では、sayHello()メソッドはどうでしょうか? ObjectAにはそのようなメソッドはありません。そこで、ObjectAの

prototypeを参照します。参照先のObjectBにはsayHello()メソッドが定義されているので、この関数が実行されます。一方、startPlay()はどうでしょうか? ObjectAにメソッドがないので、prototypeを辿ります。しかし、startPlay()はObjectBにも定義されていません。そこで、さらにprototypeを辿り、ObjectCにたどり着きます。ここでstartPlay()メソッドが見つかるので、この関数が実行されます。

3-9-3 | プロトタイプの利点

では、なぜこのようなプロトタイプという仕組みが用意されたのでしょうか? どんな利点があるのでしょうか? プロトタイプがなかった場合と比較しながらその理由を考えてみましょう。

▶過去の資産を活用できる

英語の言い回しに「Reinventing the wheel」というものがあります。「車輪の再発明」とも訳されますが、すでにあるものを作り直すのは無駄ですよね? プログラミングの世界でも、過去に作ったものは積極的に再利用することが推奨されます。関数による処理の抽象化、ライブラリの導入、オブジェクト指向言語における「継承」、これらはまさに過去の資産を活用するための仕組みです。プロトタイプも同じです。Animalという動物を表わすクラスをつくってeat()、sleep()というメソッドを定義したとします。

Animalクラスでeat()、sleep()メソッドを定義

Animal → eat
Animal → sleep

その後で、Humanという人間を表わすクラスを作る必要がでてきました。話すことは人間の特権です。よって、eat()、sleep()に加えてspeak()というメソッドが必要になりました。

これら3つのメソッドを持つHumanクラスを最初から作成することも可能です。

Humanクラスでeat()、sleep()、speak()メソッドを定義

Human → eat
Human → sleep
Human → speak

人間も動物の一種です。話すだけでなく食べたり眠ったりします。せっかくAnimalというクラスがあるのであれば、それを活用しない手はありません。以下のようにHumanオブジェクトのプロトタイプとしてAnimalを参照する方法もあります。

Humanオブジェクトのプロトタイプとして Animalを参照

プロトタイプを活用すれば、新規に実装する必要があるのはspeak()というメソッドだけになるので、おそらく実装負担は軽くなるでしょう。

▶メモリ消費量が少ない

「Hello」「World」「Welcome」という3つのStringオブジェクトを作ったとします。それぞれのオブジェクトに対してcharAt、charCodeAt、concatといったメソッドが呼び出せるようにするには、各オブジェクトにメソッドを作成・登録しなくてはなりません。オブジェクトを生成するとその分メモリが消費されます。メソッドの数が多い場合はメモリ消費量が膨大になってしまいます。

オブジェクトごとに同じメソッドをつくるのはメモリの無駄

一方、プロトタイプを使った場合、オブジェクトの様子は以下のようになります。

プロトタイプを利用するとメソッドを共用できる

プロトタイプを使ったほうが、メモリの消費量が少ないことがわかります。その効果はメソッドの数が増えるにつれ、作成するオブジェクトが増えるにつれ顕著になっていきます。

▶修正箇所を限定できる

JavaScriptでは実行中にメソッドの内容を変更することが可能です。たとえばcharCodeAt()はUnicodeの文字コードの数値を返しますが、何らかの理由でSJISの文字コード値を返すように修正したくなったと仮定しましょう。プロトタイプがなかったとすると、全部のStringオブジェクトのcharCodeAtメソッドの内容を修正する必要があります。一方、プロトタイプがある場合、修正箇所は1か所で済みます。

3-9-4 | プロトタイプの設定方法

ここまで「プロトタイプとはどのようなものか」といった概念について説明してきました。次に、JavaScriptでプロトタイプを使うには具体的にどのようにしたらよいか見ていきましょう。

ここでポイントとなるのは「プロトタイプは暗黙の参照であり、プログラマーが明示的に設定するものではない」ということです。すなわち、以下のような考え方は正しくありません。

```
var str0 = new String("Hello");
str0.prototype = someOtherObject; // prototypeを自分で明示的に設定するのは正しくない
```

確かにstr0オブジェクトにはprototypeというプロパティが追加され、その参照先がsomeOtherObjectとなります。しかし、この例でのprototypeは単なるプロパティのひとつに過ぎず、呼び出されたときに参照先を辿っていくというプロトタイプとしての働きはしてくれません。

そもそも、この例のようにオブジェクトを生成するたびにプログラマーがプロトタイプを設定するのは面倒です。プロトタイプの設定を忘れて、意図しない挙動になることもあるでしょう。ご安心ください。JavaScriptではもっとよい方法が用意されています。

JavaScriptでオブジェクトを作る際にはコンストラクタが使われます。new演算子をつけてfunctionを呼び出すと、それは関数として実行されるのではなく、オブジェクトを作成するためのコンストラクタとして動作します。そのコンストラクタに明示的にprototypeを設定しておくと、そのコンストラクタを使って作られたオブジェクトは暗黙のprototypeを参照するようになるのです。

コンストラクタを使って作られたオブジェクトはコンストラクタのprototypeを参照する

コンストラクタ —prototype→ プロトタイプオブジェクト
オブジェクト1 --prototype--> プロトタイプオブジェクト
オブジェクト2 --prototype--> プロトタイプオブジェクト
オブジェクト3 --prototype--> プロトタイプオブジェクト

具体例を見てみましょう。

```
var myProto = {
    sayHello: function () { console.log("hello") }
}                                                    ←1 プロトタイプオブジェクト

function myObject() {
}                                                    ←2 コンストラクタ

myObject.prototype = myProto;                        ←3 コンストラクタにprototypeを設定

var a = new myObject();
var b = new myObject();                              ←4 オブジェクトの作成

a.sayHello();                                        ← プロトタイプのメソッド呼び出し

var p0 = Object.getPrototypeOf(a);
var p1 = Object.getPrototypeOf(b);                   ←5
var isSame0 = (p0 === p1);                           ← p0とp1は同一(isSame0はtrue)
var isSame1 = (p0 === myProto);                      ← p0とmyProtoは同一(isSame1はtrue)
```

1のmyProtoがプロトタイプオブジェクトです。sayHelloというメソッドをひとつだけ持つシンプルなオブジェクトです。

2で定義されている関数myObjectはコンストラクタです。

3の「myObject.prototype = myProto」で、myObjectオブジェクトのプロトタイプを設定しています。

4にあるように、aとbはmyObjectのオブジェクトです。

5で、それぞれのプロトタイプオブジェクトを「Object.getPrototypeOf()」で取得しています。

aの暗黙のプロトタイプがp0で、bの暗黙のプロトタイプがp1となります。p0、p1、myProtoという3つのオブジェクトがまったく同じものであることがわかります。

▶プロトタイプの例

ゲームプログラミングをしていると、カードゲームでカードをシャッフルしたり、地雷の位置を変更したりと、配列の並び順をランダムにしたいという状況に出くわすことがあるでしょう。残念ながらもともとの配列オブジェクトArrayにそのようなメソッドは用意されていません。そのような場合に便利なメソッドを実装してみましょう。

```
<html>
    <head>
        <script>

        Array.prototype.shuffle = function () {      ←1
            var i = this.length;
            while (i) {
                var j = Math.floor(Math.random() * i);
                var t = this[--i];
                this[i] = this[j];
                this[j] = t;
            }
            return this;
        }

        function shuffle() {
            var cards = [1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6];
            cards.shuffle();    ←2
            document.getElementById("result").innerText = cards.join(",");
        }
        </script>
    </head>
    <body>
        <button onclick="shuffle()">shuffle</button>
        <p id="result"></p>
    </body>
</html>
```

メソッドの中身はFisher-Yatesというアルゴリズムを実装したものですが、今はその内容を理解しなくてもかまいません。今はプロトタイプの理解を深めることに集中しましょう。

Arrayは配列型のコンストラクタです。上記のコードでは、1で配列Arrayのコンストラクタのprototypeにshuffleというメソッドを追加しています。そうすることで、すべての配列型オブジェクトに対してshuffleメソッドを呼び出すことが可能になります。

配列Arrayのコンストラクタのprototypeにshuffleというメソッドを追加

ここでは、2でcardsという配列を定義して、cards.shuffle()とメソッドを呼びだしています。そうすると配列の要素がランダムに並べ替えられます。もともと配列型にはshuffleというメソッドはありません。しかし、これが動作するのは、cardsという配列型にshuffleというメソッドはなくても、cardsの暗黙のprototype参照の先にshuffleという関数オブジェクトが登録されているからなのです。

3-10 イベント

「JavaScriptのプログラミングはイベントハンドラを記述すること」といっても過言ではありません。それくらいイベントとイベントハンドラは重要な位置を占めます。ここでは、いろいろなイベントとそのイベントを処理する方法について説明します。

3-10-1 | イベント、イベントハンドラ

イベントとは何らかの事象のことです。人生には、誕生日、入学式、運動会、卒業式、就職といったさまざまなイベントがありますが、JavaScriptにとってのイベントは、文書が読み込まれた、マウスがクリックされた、マウスが移動した、キーが押下された、タイマーの時間が来た、といったものです。JavaScriptのプログラミングとは「イベントが発生したときにどのような処理を行うか記述すること」といっても過言ではありません。それだけに、イベントについて正しい理解を持つことはとても重要です。

▶イベントハンドラ

イベントが発生したときに実行される関数を「イベントハンドラ」と呼びます。たとえば、誕生日というイベントにとっては、誕生日パーティーがイベントハンドラに該当するでしょう。JavaScriptでは「プロパティに関数を代入する」という方法でイベントハンドラを設定します。言葉による説明よりも例を見たほうが直感的にわかりやすいと思いますので、早速具体例を見てみましょう。

3-10-2 | 文書の読み込みイベント

人生最初のイベントが誕生であるように、JavaScriptの最初のイベントは文書読み込みのイベントです。このイベントの処理方法を見てみましょう。

単純なHTML SAMPLE eventload0.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
</head>
<body>
    <h1>こんにちは</h1>
</body>
</html>
```

ブラウザ表示例

D:¥Book¥JavaScript¥Eve

こんにちは

イベントハンドラの使用例

SAMPLE eventload1.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <script>
window.onload = init;    ←1

function init() {
    var h = document.getElementById("title");
    h.textContent = "はじめまして";                ←2
}
    </script>
</head>
<body>
    <h1 id="title">こんにちは</h1>
</body>
</html>
```

ブラウザ表示例

D:¥Book¥JavaScript¥Eve

はじめまして

1で、windowオブジェクトのonloadプロパティにイベントハンドラinitを登録しています。こうすると、文書の読み込み完了後に関数initが呼び出されます。2のイベントハンドラ関数initの中では、「id="title"」という要素を取得し、そのtextContentプロパティに文字列を代入することで表示を「はじめまして」に更新しています。ちなみに、以下のように無名関数を使うことも可能です。

SAMPLE eventload2.html

```
window.onload = function () {
    var h = document.getElementById("title");
    h.textContent = "はじめまして";
};
```

また、以下のようにbody要素のonload属性に記述してもほぼ同じ挙動（文書読み込み後に関数initが実行される）となります。

```
<body onload="init()">
```

昔はbody要素のonload属性にイベントハンドラを記述するのが一般的でしたが、最近ではwindowオブジェクトのonloadハンドラに登録する手法や、以下のようにDOMContentLoadedイベントにハンドラを登録する手法も見受けられるようになりました。

SAMPLE eventload3.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <script>
    document.addEventListener('DOMContentLoaded', function () {
        var h = document.getElementById("title");
        h.textContent = "はじめまして";
    });
    </script>
</head>
<body>
    <h1 id="title">こんにちは</h1>
</body>
</html>
```

いずれも、文書を読み込み時のイベントを処理するものです。ただ、厳密にいうとwindowオブジェクトのonloadとdocumentオブジェクトのDOMContentLoadedイベントは発生するタイミングが異なります。

DOMContentLoadedとonload

DOMContentLoaded
onload
画像などの読み込み
読み込み開始
HTML文書の読み込み・解釈完了
文書全体の読み込み完了

ブラウザはHTMLを読み込むと、先ず文書がどのような構造かを解釈をします。この解釈が完了した段階でDOMContentLoadedが発生し、その後で画像などの読み込みが行われます。画像などの読み込みがすべて完了した時点でonloadが発生します。実際に試してみましょう。<script>要素に以下のように記述してみます。

SAMPLE eventload4.html

```
document.addEventListener('DOMContentLoaded', function () {
    console.log("called: DOMContentLoaded");
});
window.onload = function () {
    console.log("called: window.onload");
}
```

開発者ツールのコンソールを見てみると、DOMContentLoadedが先に呼び出されていることがわかります。

デバッガのコンソールで確認

event4.html - F12 開発者ツール
F12　DOM Explorer　コンソール　デバッガー　Edge
0　1　1　ターゲット: _top: event4.html
HTML1300: ナビゲーションが発生しました。
ファイル: event4.html
called: DOMContentLoaded
called: window.onload

画像の読み込みなどに時間がかかるページで、早い段階でJavaScriptの処理を行いたい場合はDOMContentLoadedを、特にこだわりがない場合は好きなほうを使えばよいでしょう。

(3-10-3 | ボタンのクリック)

次にボタンをクリックしたときに発生するイベントハンドラを見てみましょう。

SAMPLE mouseclick0.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <script>
window.onload = function () {
    document.getElementById("yes").onclick = yeshandler;
    document.getElementById("no").onclick = nohandler;
}
function yeshandler(e) {
    document.getElementById("status").textContent = "はいが選ばれました"
}
function nohandler(e) {
    document.getElementById("status").textContent = "いいえが選ばれました"
}
```

```
    </script>
</head>
<body>
    <button id="yes">はい</button>
    <button id="no">いいえ</button>
    <h2 id="status"></h2>
</body>
</html>
```

ブラウザ表示例

「はい」「いいえ」、ボタンを押下すると画面の表示が更新されます。window.onloadでは、それぞれのボタンをdocument.getElementById()で取得し、そのonclickプロパティに専用のハンドラを登録しています。その様子を以下の図に示します。

各ボタンにハンドラを登録

このアプローチでは、それぞれのイベントに専用のイベントハンドラを割り当てるので直感的でわかりやすいと思います。しかし、どちらのハンドラでも同じような処理を行うので、都度記述するのは無駄ですよね。以下のようにすればひとつのハンドラで同じ効果を実現できます。

```
window.onload = function () {
    document.getElementById("yes").onclick = myhandler;
    document.getElementById("no").onclick = myhandler;
}
function myhandler(e) {
    document.getElementById("status").textContent =
        e.target.textContent + "が選ばれました"
}
```

ブラウザ表示例

はい　いいえ

いいえが選ばれました

onclick

onclick

function myhandler(e)

なぜひとつのハンドラでそれぞれのボタンに対応できたのでしょうか?　ここで注目してほしいのは、myhandler()の中にあるe.targetです。変数eはイベントハンドラが呼び出されるときの引数で、ブラウザが値を設定してくれます。変数eのtargetプロパティにはイベントを発生させた要素、すなわちbutton要素への参照が格納されます。つまり、このプロパティをみればどのボタンが押下されたかがわかるのです。あとは、要素のtextContentプロパティからボタンのラベル文字列を取得し、その内容を設定しています。

3-10-4 | イベントハンドラの引数

上記では

- イベントハンドラが呼ばれるときにはブラウザが引数をセットしてくれる
- その引数のtargetプロパティをみるとイベントを発生させた要素がわかる

と説明しました。しかし、一口に、イベントといっても多くのバリエーションがあり、それぞれに付随する情報は異なります。たとえば、input要素でスライダバーが操作されたときは、値が変化した旨の情報があれば十分ですが、canvas要素などでマウスが操作されたのであれば座標情報も必要となるでしょう。少し実験をしてみましょう。

SAMPLE eventarg1.html

```
<!DOCTYPE html>
<html>
<head>
    <script>
window.onload = function () {
    var nodes = document.getElementsByClassName("target");
    for (var i = 0 ; i < nodes.length ; i++) {
        nodes[i].onclick = myhandler;
        nodes[i].onmousedown = myhandler;
        nodes[i].onmouseup = myhandler;
        nodes[i].onchange = myhandler;
        nodes[i].onfocus = myhandler;
    }
}
```

```
function myhandler(e) {
    console.log(e.type + ": tagName=" + e.srcElement.tagName);
    console.log("  clientX=" + e.clientX + " screenX=" + e.screenX + " pageX=" + e.pageX);
}
    </script>
</head>
<body>
    <button class="target">ボタン</button>
    <input class="target" type="range"/>
    <p class="target">パラグラフ</p>
    <h3>デベロッパーツール（デバッガ）を起動してコンソール出力をご覧ください</h3>
</body>
</html>
```

ブラウザ表示例

デバッガのコンソール

button要素、input要素、p要素に対して、onclick、onmousedown、onmouseup、onchange、onfocusといったイベントのハンドラを設定しています。イベントハンドラmyhandlerでは、引数で与えられたイベントに関する情報をコンソールに出力しています。

初期化関数onloadの中にdocument.getElementsByClassNameという見慣れない命令があります。このgetElementsByClassNameは、引数で指定されたクラス名を持つ要素をまとめて返す関数です。document.getElementByIdは引数で指定されたidを持つ要素を返す関数でした。これと働きは似ています。ただ、idは文書中で同じ値を複数指定できませんでしたが、class属性には同じ値を割り当てられるので、複数の要素を抽出できます。覚えておくと便利です。

コンソール出力をみると、どのようなイベントが発生するかがわかります。発生する順序や内容はブラウザによって異なりますが、さまざまなイベントが発生していることが確認できます。IEの場合は以下のようなイベントが発生していました。

イベントの種類と発生順

- ボタンの押下　mousedown → focus → mouseup → click
- スライダバーの操作　mousedown → change → change → focus → mouseup → click
- パラグラフのクリック　mousedown → mouseup → click

また、focus、changeといったイベントのときはclientX, screenX, pageXといったマウス座標に関連する情報が設定されていない（undefined）こともわかります。

このようにイベントハンドラに渡された引数を参照することで、どの要素でどのような操作が行われたか、マウスの場合はどの座標でイベントが発生したのか、ということがわかるようになります。

ところで、前の例ではマウスカーソルの座標値を取得するために、clientX, screenX, pageXといったプロパティを使用していました。これらはX座標（横方向）の値を取得するためのものですが、Y座標の値を取得するためにclientY, screenY, pageYといったプロパティも用意されています。では、これらのプロパティはいったい何が違うのでしょうか？ 以下の図をご覧ください。

マウスカーソルの座標値

ページの原点 pageX/Y
スクリーンの原点
screenX/Y
ブラウザ領域の原点
clientX/Y
自要素の原点
offsetX/Y
ページの範囲
clientX=114, clientY=109
pageX=114,
pageY=149.25999450683594
screenX=245, screenY=254
offsetX=56,
offsetY=17.020004272460937

それぞれのプロパティは原点とする座標が異なります。

プロパティと座標

プロパティ	座標
screenX/Y	画面（スクリーン）の左上を原点とする座標
clientX/Y	ブラウザ領域の左上を原点とする座標
pageX/Y	ページの左上を原点とする座標（スクロールしていないときはclientX/Yと同じ値）
offsetX/Y	イベント発生元の要素（自要素）の左上を原点とする座標

Canvasでゲームを実装するときは、原点をCanvasの左上端とするoffsetX/Yを使うことが多いでしょう。こうすればCanvasをページ上のどこに配置してもJavaScriptのコードを変更する必要がありません。しかし、

Firefoxは現時点においてoffsetX/Yをサポートしていないようです（http://www.jacklmoore.com/notes/mouse-position/）。そこで本書では以下のような回避策を使用することにしました。ただし、紙面の都合上、本書のすべてのサンプルでこのような記載をしているとは限りません。その旨ご了承ください。

SAMPLE eventarg2.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <style>
        p {
            background-color:gray;
            width:400px; height:400px;
            padding:50px; font-size:24px;
        }
    </style>
    <script>
        window.onload = function () {
            document.getElementById("area").onmousemove = mymousemove;
        }
        function mymousemove(e) {
            document.getElementById("s0").textContent =
                "clientX=" + e.clientX + ", clientY=" + e.clientY;
            document.getElementById("s1").textContent =
                "pageX=" + e.pageX + ", pageY=" + e.pageY;
            document.getElementById("s2").textContent =
                "screenX=" + e.screenX + ", screenY=" + e.screenY;
            var target = e.target || e.srcElement,
                    rect = target.getBoundingClientRect(),
                    offsetX = e.clientX - rect.left,          ←1
                    offsetY = e.clientY - rect.top;
            document.getElementById("s3").textContent =
                "offsetX=" + offsetX + ", offsetY=" + offsetY;
        }
    </script>
</head>
<body>
<h1>マウスイベントのプロパティ</h1>
    <p id="area">
        <span id="s0"></span><br />
        <span id="s1"></span><br />
        <span id="s2"></span><br />
        <span id="s3"></span><br />
    </p>
</body>
</html>
```

1の部分が今回の対処策になります。まず、引数eのtargetプロパティを取得します。ブラウザによってはtargetがない場合がありますが、その際はsrcElementプロパティで代用します。

あとは、getBoundingClientRectで自分の領域を求め、clientX/Yから自領域の左rect.left、上rect.topの差分を求めて、最終的にoffsetXとoffsetYを求めます。

少々内容が難しいので現時点では理解できなくても大丈夫です。ただ、イベントハンドラの引数で得られる座標値にはさまざまなものがあるので注意が必要なことは覚えておいてください。

HTMLやCSSの仕様はW3Cなどで細かく規定されています。ただ、ブラウザによって対応状況も異なりますし、挙動にも違いがあります。特にイベント回りは違いが大きい部分です。実際にはいろいろなブラウザで試行錯誤しながら、プロパティやイベントを探していく作業が必要でしょう。

3-10-5 | イベントハンドラの登録先

ボタンを押したときの処理は、各ボタンごとにイベントハンドラを記述するか、複数のボタンをまとめて処理するか、場合によりけりです。例として、3つのカードからあたりをひとつ当てる簡単なゲームを紹介しましょう。まず、個々の要素にイベントハンドラを割り当てる方法です。

SAMPLE event-scope0.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <style>
        .card {
            width: 50px;
            height: 70px;
            border: 1px solid blue;
            border-radius: 10px;
            text-align: center;
            font-size: 26px;
            background-color: white;
            box-shadow: rgb(128, 128, 128) 5px 5px;
        }
    </style>
    <script>
        var strike = Math.floor(Math.random() * 3);          ←1
        window.onload = function () {
            document.getElementById("card0").onclick = myhandler0;
            document.getElementById("card1").onclick = myhandler1;
            document.getElementById("card2").onclick = myhandler2;      ←2
            document.getElementById("shuffle").onclick = shuffle;
        }
```

```
        function myhandler0(e) {
            if (strike == 0) {
                document.getElementById("card0").textContent = "○";
            }
        }
        function myhandler1(e) {
            if (strike == 1) {
                document.getElementById("card1").textContent = "○";     ←3
            }
        }
        function myhandler2(e) {
            if (strike == 2) {
                document.getElementById("card2").textContent = "○";
            }
        }
        function shuffle(e) {
            strike = Math.floor(Math.random() * 3);
            document.getElementById("card0").textContent = "";
            document.getElementById("card1").textContent = "";          ←4
            document.getElementById("card2").textContent = "";
        }
    </script>
</head>
<body>
    <div id="deck">
        <table>
            <tr>
            <td class="card" id="card0"> </td>
            <td class="card" id="card1"> </td>
            <td class="card" id="card2"> </td>
            </tr>
        </table>
        <button id="shuffle">Shuffle</button>
    </div>
</body>
</html>
```

コードは少々長めですが処理内容はシンプルです。2のonloadで、それぞれのカードとボタンにイベントハンドラを登録しています。

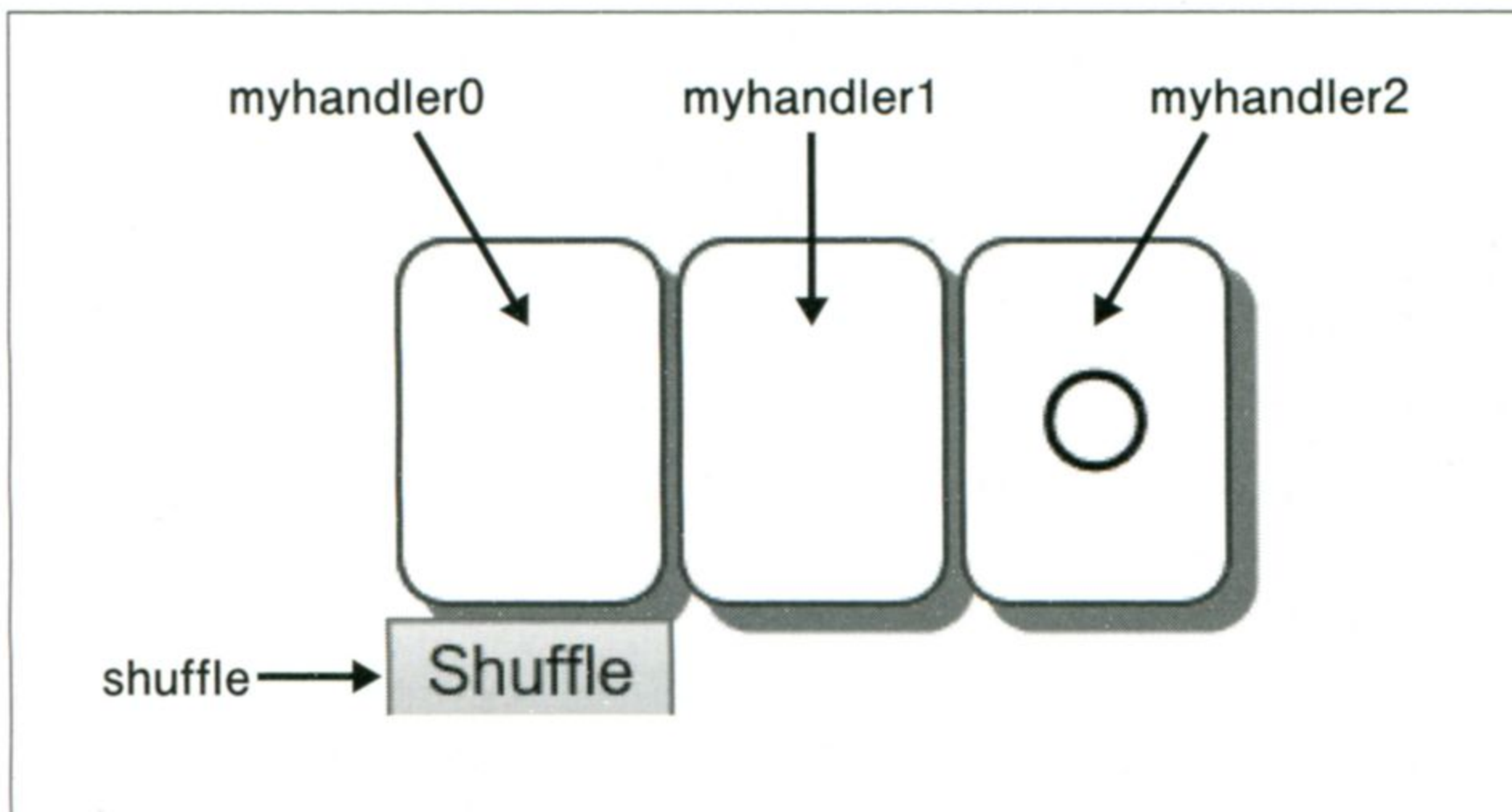
また、1で当たりカードの番号（0～2）を乱数で生成し、変数strikeに設定しています。

3では、それぞれのイベントハンドラに対し、自分のカードと当たり番号が同じだったときに「○」を表示しています。

4のshuffle関数は、Shuffleボタンが押下されたときのイベントハンドラで、乱数を設定し、すべてのカード

を裏返しにしています。

ここで、すべてのカード、Shuffleボタンに別々のイベントハンドラを登録していることに注目してください。

すべてのカード、Shuffleボタンに別々のイベントハンドラを登録

一方、以下のサンプルは動作としてはまったく同じですが、イベントハンドラの登録対象が異なります。

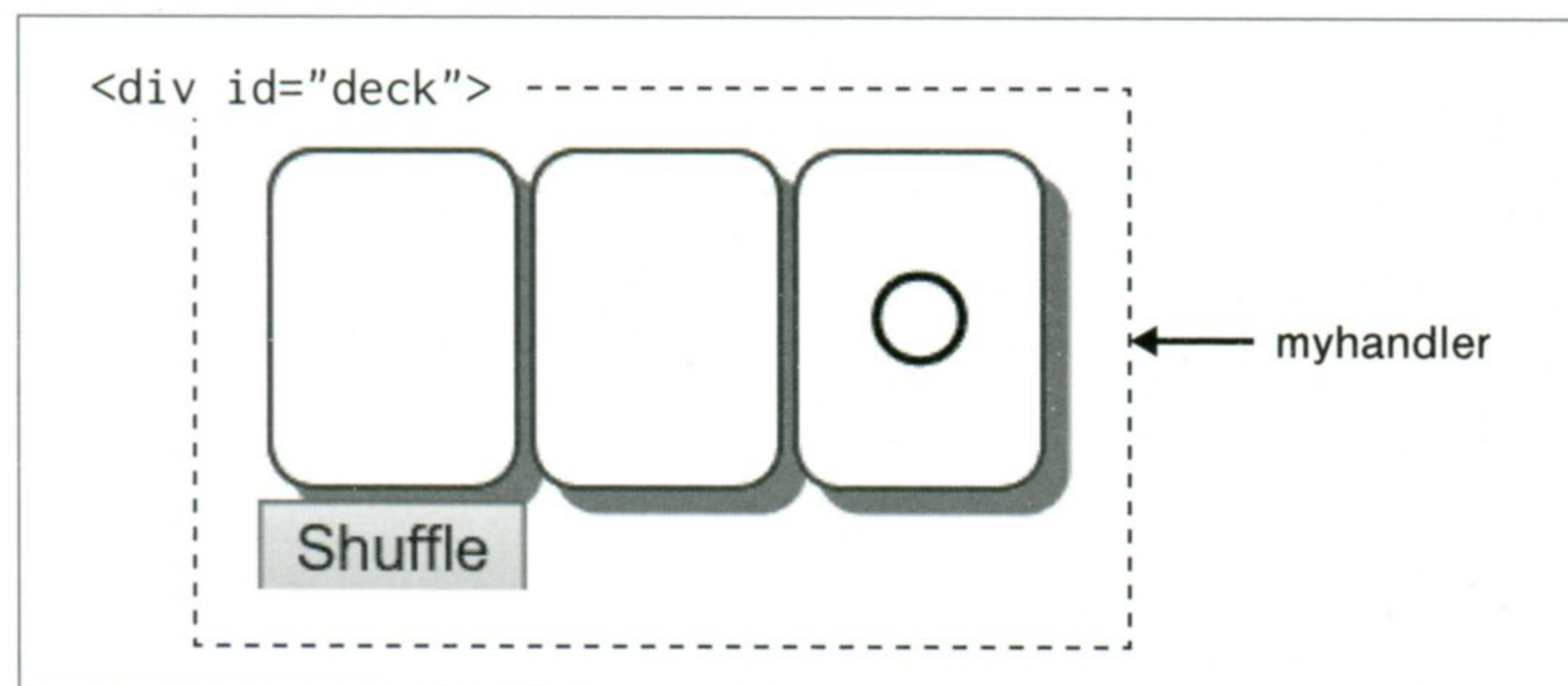
SAMPLE event-scope1.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <style>
        .card {
            width: 50px;
            height: 70px;
            border: 1px solid blue;
            border-radius: 10px;
            text-align: center;
            font-size: 26px;
            background-color: white;
            box-shadow: rgb(128, 128, 128) 5px 5px;
        }
    </style>
    <script>
        var strike = Math.floor(Math.random() * 3);
        window.onload = function () {
            document.getElementById("deck").onclick = myhandler;      ←1
        }

        function myhandler(e) {
            var card0 = document.getElementById("card0");
            var card1 = document.getElementById("card1");
            var card2 = document.getElementById("card2");
            var shuffle = document.getElementById("shuffle");
```

```
            if (e.target == shuffle) {
                strike = Math.floor(Math.random() * 3);
                card0.textContent = "";
                card1.textContent = "";
                card2.textContent = "";
            }
            if (e.target == card0 && strike == 0 ||
                e.target == card1 && strike == 1 ||
                e.target == card2 && strike == 2) {
                e.target.textContent = "○"
            }
        }
    </script>
</head>
<body>
    <div id="deck">
        <table>
            <tr>
            <td class="card" id="card0"> </td>
            <td class="card" id="card1"> </td>
            <td class="card" id="card2"> </td>
            </tr>
        </table>
        <button id="shuffle">Shuffle</button>
    </div>
</body>
</html>
```

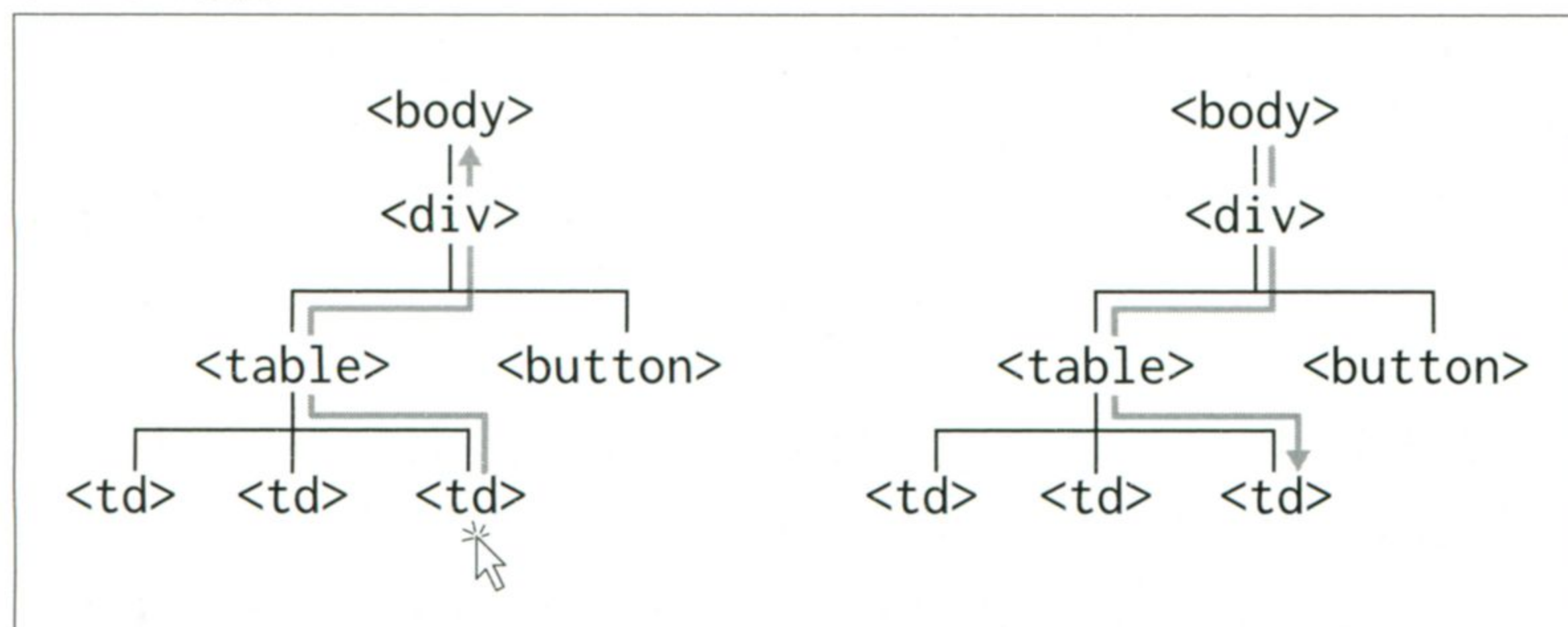
イベントハンドラはdiv要素にひとつ登録しているだけです1。イメージを以下に示します。イベントハンドラmyhandlerの中では、イベントを発生した要素e.targetに応じて処理を切り分けています。

イベントハンドラはdiv要素にひとつだけ登録

<div id="deck">
○
myhandler
Shuffle

このように、同じ目的を達成するにしてもいろいろな方法があることがわかります。どちらがよい悪いという問題ではなく、状況に応じて使い分けられるようになるのが理想だと思います。

▶イベントの通知

先の例では、親要素でまとめてイベントを受信しましたが、JavaScriptでは、子要素→親要素、親要素→子要素のどちらの順番でイベントが通知されていくのかを制御することができます。

イベントの通知

```
        <body>                        <body>
          ↑                             ||
        <div>                         <div>

  <table>     <button>          <table>     <button>

<td>  <td>  <td>             <td>  <td>  <td>
```

今回は、以下のようにイベントハンドラを登録しました。

```
document.getElementById("deck").onclick = myhandler;
```

この場合は、上左図に示されるように、子要素→親要素の順番にイベントが通知されていきます。泡がのぼっていくようにも見えるのでEvent Bubblingとも呼ばれます。

一方、親要素から順番に要素までイベントが通知されていく様子をCapturingと呼びます。この順番にイベントを処理したい場合は、

```
elements[i].addEventListener("click", myhandler, true);
```

のように記述します。ただ、Bubbling/Capturingはすべてのブラウザが対応しているとは限りません。どちらの順番で通知が行われたとしても動作するような作りにしておくほうがよいでしょう。

NOTE　Bubbling, Capturingの動作確認用にevent-scope2.htmlを用意しました。興味のある方は中身をご覧ください。

SAMPLE　event-scope2.html

3-10-6 | タッチイベントに関して

タブレットやスマートフォンの場合、マウスやキーボードではなくタッチを使用します。本書でもタッチを意識してサンプルを作成しましたが、ブラウザによって挙動にばらつきがあり、すべてのブラウザに対応することは困難でした。読者ができる限りコードを改変せずに、多くのOS/ブラウザに対応できるようにしましたが、すべてのOS/ブラウザに対応しているわけではない旨ご了承ください。

「マウスポインタを指に置き換えるだけだから簡単なのでは?」と思う方もいるかもしれません。しかしながら、以下のような事項を考えると、一筋縄ではいかないことがわかると思います。

- マウスクリックは1点のみだがタッチは指の数だけ接点があり、それを考慮する必要がある
- 2本の指でズームや回転を行うジェスチャーという操作も考慮する必要がある
- タッチデバイスは急速に普及したので標準化が追いついていない、もしくはブラウザの実装が追いついていない

このような状況なので、どのブラウザでも同じように動作するにはまだしばらく時間がかかりそうです。ただし、どうしてもタッチをサポートしたい場合は、特定のブラウザ、OSに特化した実装にすればそれほど難しいものでもありません。

本書の後半ではゲームの内容について解説していますが、本書におけるタッチへの対応方法について以下に説明します。

- 「#canvas { touch-action: none;}」のようにタッチ対象の要素にtouch-actionプロパティを設定し、デフォルトのタッチジェスチャー操作を無効にする
- マウスのイベントハンドラに加え、タッチ用のイベントハンドラも追加する。

```
canvas.onmousedown = mymousedown;
canvas.onmousemove = mymousemove;
canvas.onmouseup = mymouseup;
canvas.addEventListener('touchstart', mymousedown);
canvas.addEventListener('touchmove', mymousemove);
canvas.addEventListener('touchend', mymouseup);
```

これらのイベントハンドラの中では、イベントの種類に応じて利用できるプロパティが異なるため、以下のようにプロパティが取得できなかった場合のバックアップも指定します。ちなみにtouchesはタッチしている点の配列です。

```
function mymousedown(evt) {
    var mouseX = !isNaN(e.offsetX) ? e.offsetX : e.touches[0].clientX;
    var mouseY = !isNaN(e.offsetY) ? e.offsetY : e.touches[0].clientY;
```

isNaN()は引数が数字でないか否かを調べる関数です。「Is Not A Number」という英語が関数名の由来です。仮にマウスで操作している場合であれば、e.offsetXに数値が格納されているはずです。その場合は「!isNaN(e.offsetX)」がtrueになるのでe.offsetXの値を返します。数値でない場合はタッチ操作が行われたとして、e.touches[0].clientXの値を返しています。

Windows10のChromeで試したところ'touchend'のコールバックでは接点の数が0になるためかevt.touchesが空の配列になっていました。このような挙動も正しく認識したうえで使用する必要があることに注意してください。

また、この方法はあくまでも暫定的なものであり、すべてのブラウザでの動作を保証するものではありません。現在の状況を見ると、ブラウザの対応状況をJavaScriptから調べてそれに応じて適切な処理を行う、もしくはそのような差異を吸収してくれるライブラリを使用するのが妥当なアプローチとなるでしょう。

3-11 関数オブジェクト

本章の最後は関数オブジェクトです。関数をモノとして扱う方法について勉強します。直観的ではないため最初は違和感を覚えるかもしれませんが、ワンランク上を目指すには必須のテクニックです。

3-11-1 | 関数はオブジェクト

実は、JavaScriptでは関数もオブジェクトです。こう聞くと違和感を覚えるかもしれません。「JavaScriptの関数は、何かしら決まった処理を行うものでしょ？ 何でこれがオブジェクトなの?!」そう感じるのは自然な反応だと思います。ただ、JavaScriptでワンランク上を目指すのであれば関数オブジェクトは避けて通れません。本節の目的は“関数=オブジェクト”という感覚に慣れていただくことです。

まずは、身近な例を使って説明してみましょう。料理のレシピを考えてみてください。

レシピとは、「どんな材料を用意して、どんな手順で調理すれば、どんな料理が完成するか記載したもの」です。レシピであれば見たり触ったりできるのでオブジェクト（モノ）っぽいですよね。

では、関数をこんなふうに考えてみてください。関数とは、「どんな引数を受領して、どんな手順で処理すれば、どんな出力が得られるか記述したもの」です。レシピと対比すると少しは関数もオブジェクト（モノ）っぽく感じられるのではないでしょうか?

実際に、JavaScriptでは「関数は処理手順を定義したオブジェクト（モノ）」として扱われます。オブジェクトなので変数に代入したり、引数として別の関数に渡したりすることもできます。関数オブジェクトがほかのオブジェクトや変数と大きく異なるのは「()」を後ろにつけることで実行できることです。

SAMPLE funcobj0.html

```
function add(a, b) {
    return a + b;
}
function mul(a, b) {
    return a * b;
}                                    ←1
var calculator = add;
var x = calculator(2, 3);   // x=5   ←2
calculator = mul;
var y = calculator(2, 3);   // y=6   ←3
```

1でふたつの関数add, mulを宣言しています。関数はオブジェクトなので変数に代入することができます。まず2でaddを変数calculatorに代入し、calculatorに「(2, 3)」とつけることで関数を実行しています。これによりaddが実行され、xには5が代入されます。

次に**3**でcalculatorにmulを代入して、同様に実行すると、今度はmulが実行され、yには6が格納されます。

以下のように、関数の宣言と変数への代入を同時に行うことも可能です。

SAMPLE funcobj0.html

```
var calculator = function (a, b) {
    return a / b;
}
var z = calculator(3, 2);   // z=1.5
```

上記は割り算を実行する関数ですが、関数名は明示的に記述していません。このように関数名を明示的に指定しないものを「無名関数オブジェクト」と呼びます。ちなみに、「var calculator = function div(a, b) {…」のように関数名を明示的に記述しても問題はありません。

無名関数はオブジェクトのメソッドを定義するときによく使われます。

SAMPLE funcobj0.html

```
function Person(name, energy) {
    this.name = name;
    this.energy = energy;
    this.run = function () { this.energy -= 10; }
    this.sayHello = function () { console.log("Hi! " + this.name); }
}

var p = new Person("Tanaka", 100);
p.run();          // energyが90に減る
p.sayHello();    // consoleに"Hi! Tanaka"と表示される
```

関数Personはコンストラクタで、Personクラスのオブジェクトを作成するために使用されます。このPersonクラスには、name(文字列)、energy(数値)、run(関数)、sayHello(関数)といったプロパティが登録されています。言語によっては、メンバ変数（文字列・数値・オブジェクトなどの値）と、メソッド（関数）を明確に区別するものもありますが、JavaScriptでは、どちらも同じプロパティです。その参照先が、数値や文字といった変数であればメンバ変数になり、参照先が関数オブジェクトであればメソッドになるといった違いでしかありません。

参照先が数値や文字であればメンバ変数、関数オブジェクトであればメソッド

Personオブジェクト		
name	→	"Tanaka"
energy	→	100
run	→	function
sayHello	→	function

関数を変数に格納したり、プロパティとして宣言したり、といった特徴は文字列や数値とまったく同じです。「関数も単なるオブジェクトに過ぎない」という考え方にも慣れてきたのではないでしょうか? そうでない人は「関数はオブジェクト(モノ)である」と何度か自分に言い聞かせましょう。今の時点では違和感があったとしても、そのうち慣れるので安心してください。

関数をオブジェクトにして扱う理由

ところで、「関数をオブジェクトにして何が嬉しいの?」「何のために変数に代入する必要があるの」と釈然としない人も多いのではないでしょうか。ここで、関数をオブジェクトにして扱うという理由について考えてみましょう。

例を使って説明しましょう。携帯ゲーム機を想像してください。携帯ゲーム機ではカートリッジやメモリカードを挿入して遊びます。カートリッジにはどんな処理を行うかが記録されていますが、カートリッジだけではまったく役に立ちません。実行する本体があって初めて価値があるのです。カートリッジを差し替えれば別のゲームも遊べます。

関数オブジェクトはカートリッジやメモリカードに相当します。関数オブジェクトにはどんな処理を行うかが記述されていますが、関数オブジェクト単体ではまったく役に立ちません。それを利用する呼び出し側があって初めて意味があるのです。関数オブジェクトを別のオブジェクトに取り換えるとまったく違う処理を実行できるようになります。

このように関数オブジェクトの存在意義を理解するためには、関数オブジェクト単体ではなく、呼び出し側とのペアで考えることがポイントです。

(3-11-2 | 関数オブジェクトによる配列の操作)

関数オブジェクトの習得は場数をこなして慣れるのが一番です。配列の要素すべてを合計する処理を考えてみましょう。for 文を使うと以下のようになります。

```
var data = [1, 6, 3, 4, 3, 2, 6, 8, 5, 9, 0];
var total = 0;
for (var i = 0 ; i < data.length ; i++) {
    total += v;
}
```

関数オブジェクトを使うとよりシンプルに記述できます。

```
var data = [1, 6, 3, 4, 3, 2, 6, 8, 5, 9, 0];   ←1
var total = 0;
data.forEach(function (v) { total += v });   ←3
             ─────────────2─────────────
```

2の「function (v) { total += v }」の部分が関数オブジェクトです。どちらもtotalの値は同じになります。

1のdataはArrayオブジェクト（配列）なので、dataに対してArrayオブジェクトのメソッドを呼び出すことができます。

3のforEachはArrayオブジェクトのメソッドのひとつで、配列の要素を順番に取り出し、その値を引数として関数オブジェクトを実行します。この例では、

- data[0]は1 ➡ `function (1) { total += 1}`
- data[1]は6 ➡ `function (6) { total += 6}`
- data[2]は3 ➡ `function (3) { total += 3}`

  …

のように関数オブジェクトが順番に実行され、合計値totalを求めています。

関数オブジェクトの部分を書き換えると、別の処理が行えます。たとえば、

```
data.forEach(function (v) { total *= v });
```

とすれば、すべての要素を掛け合わせた数値が得られます。ちなみに、以下のように記述することも可能です。

```
var adder = function (v) { total += v };
data.forEach(adder);
```

関数オブジェクトを変数adderに格納し、それをforEachに渡しています。処理はまったく同じです。

別の例を見てみましょう。ある配列から偶数のみを取り出す処理を考えてみます。for文を使って書くと、たとえば以下のようになります。

```
var data = [1, 6, 3, 4, 3, 2, 6, 8, 5, 9, 0];
var result = [];
for (var i = 0 ; i < data.length ; i++) {
    if (data[i] % 2 == 0) {
        result.push(data[i]);
    }
}
```

これを関数オブジェクトで書き直すと以下のようになります。

```
var data = [1, 6, 3, 4, 3, 2, 6, 8, 5, 9, 0];
var result = data.filter(function (v) { return v % 2 == 0 });
                                  1
```

1の「function (v) { return v % 2 == 0 }」の部分が関数オブジェクトです。filterメソッドもforEachと同じくArrayオブジェクトのメソッドで、配列の各要素をとりだし、それを引数として関数オブジェクトを実行します。ただ、forEachと異なるのは、最終的に、関数オブジェクトがtrueを返した要素のみを含む配列を返すことです。

たとえば、vに4が渡されたときは「4 % 2 == 0」がtrueとなるため、4はresultに含まれます。一方、vに5が渡されたときは「5 % 2 == 0」はfalseとなるため、5はresultに含まれなくなります。つまり、「{return v % 2 == 0}」といった条件を満たす要素だけを抽出することができるのです。

3-11-3 | 関数オブジェクトを引数にとるArrayのメソッド

「3-8-3 Arrayオブジェクト」(P.123)ではArrayオブジェクトのメソッドについて説明しましたが、最近の仕様改定でArrayオブジェクトに便利なメソッドが追加されました。その多くが関数オブジェクトを効果的に活用するものなので、ここでまとめて紹介しておきます。forEach、filterについてはすでに具体例を示しましたが、復習もかねてもう一度説明します。

Array.prototype.forEach (関数オブジェクト)

配列の各要素に対して関数オブジェクトを呼び出します。

```
var data = [2, 5, 4, 7, 1, 6];
data.forEach(function (e, i) {
    console.log("[" + i + "]=" + e);
});
```

コンソール出力は次のようになります。配列の個々の要素について関数オブジェクトが呼び出されていることがわかります。

```
[0]=2
[1]=5
[2]=4
[3]=7
[4]=1
[5]=6
```

慣れ親しんだfor文でも同じ処理が可能です。

```
for (var i = 0 ; i < data.length ; i++) {
    console.log("[" + i + "]=" + data[i]);
}
```

ただ、forEachとfor文では厳密にいうと挙動が若干異なります。配列の一部要素をdelete命令で削除した後で、実行してみます。

forEach文

```
var data = [2, 5, 4, 7, 1, 6];
delete data[3]; ⬅ 3番目の要素「7」を削除
data.forEach(function (e, i) {
    console.log("[" + i + "]=" + e);
});
```

コンソールの出力

```
[0]=2
[1]=5
[2]=4
[4]=1
[5]=6
```

for文

```
var data = [2, 5, 4, 7, 1, 6];
delete data[3]; ← 3番目の要素「7」を削除
for (var i = 0 ; i < data.length ; i++) {
    console.log("[" + i + "]=" + data[i]);
}
```

コンソールの出力

```
[0]=2
[1]=5
[2]=4
[3]=undefined
[4]=1
[5]=6
```

forEachを使った方法の場合、削除した要素について関数オブジェクトは実行されません。一方、for文は要素の有無はおかまいなしで順番にアクセスしていきます。処理速度も若干違うようですが、これはブラウザの実装によって変わってくるので優劣はつけ難いと思います。どちらか好きなほうを使えばよいでしょう。

Array.prototype.every (関数オブジェクト)

配列のすべての要素がある条件を満たしているときにtrueを返し、ひとつでも条件を満たさない場合はfalseを返します。forEachと同様に、配列の個々の要素に対して関数オブジェクトが呼び出されます。この関数オブジェクトで「条件を満たしている場合にtrueを、満たしていない場合にfalseを返す」ように記述します。everyメソッドは個々の要素をチェックしているときに、条件を満たさないものが見つかった場合、ただちに処理を中断してfalseを返します。

```
var data = [2, 5, 4, 7, 1, 6];
var r0 = data.every(function (e) { return e < 6 });     // r0 = false
var r1 = data.every(function (e) { return e < 10 });    // r1 = true
var r2 = data.every(function (e) { return e > 0 });     // r2 = true
```

最初の例では、要素7が6より大きく、「e < 6」がfalseになるため、メソッドの戻り値r0もfalseとなります。2番目の例は、どの要素も10より小さいのでtrueが、3番目の例は、どの要素も0より大きいのでtrueが返されます。

Array.prototype.some (関数オブジェクト)

everyとは対照的に、どれかひとつでも要素が条件を満たしているときにtrueを返し、すべての要素が条件を満たさない場合にfalseを返します。forEachと同様に、配列の個々の要素に対して関数オブジェクトが呼

び出されます。この関数オブジェクトで「条件を満たしている場合にtrueを、満たしていない場合にfalseを返す」ように記述します。someメソッドは個々の要素をチェックしているときに、どれか1つでも条件を満たすものが見つかった場合、ただちに処理を中断してtrueを返します。

```
var data = [2, 5, 4, 7, 1, 6];
var r0 = data.some(function (e) { return e < 6 });     // r0 = true
var r1 = data.some(function (e) { return e > 7 });     // r1 = false
var r2 = data.some(function (e) { return e > 6 });     // r2 = true
```

最初の例では、先頭の要素2は6よりも小さく、「e < 6」がtrueになるため、メソッドの戻り値r0もtrueとなります。2番目の例は、7よりも大きい要素はないためfalseが、3番目の例は、「e > 6」を満たす要素7が存在するため、trueが返されます。

Array.prototype.filter (関数オブジェクト)

条件を満たす要素のみを含む新しい配列を返します。配列の要素ごとに関数オブジェクトを呼び出し、この関数オブジェクトがtrueを返した要素のみが新しい配列に含まれます。

```
var data = [2, 5, 4, 7, 1, 6];
var r0 = data.filter(function (e) { return e % 2 == 0 });   // r0 = [2,4,6]
var r1 = data.filter(function (e) { return e % 2 == 1 });   // r1 = [5,7,1]
var r2 = data.filter(function (e) { return e <= 4 });       // r2 = [2,4,1]
```

最初の例では、「e % 2 == 0」を満たすのは偶数だけなので、結果として得られる配列は「[2,4,6]」となります。次の例は逆に奇数だけが条件を満たすので、結果として得られる配列は「[5,7,1]」となります。3番目の例では、「e <= 4」を満たすもの、すなわち4以下の要素からなる配列「[2,4,1]」が結果として返されます。

Array.prototype.sort(比較用の関数オブジェクト)

Arrayクラスには、その設計当初から要素を並べ替えるためのsortメソッドが用意されています。言語を設計した人が並べ替え用のメソッドを事前に準備してくれているのは非常にありがたいことです。もし用意されていなかったら、クイックソート、バブルソートなどを使って並べ替え用の処理を自分で実装しなくてはなりません。しかしながら、以下のような疑問が出てくると思います。

「言語を設計した人は、私がどのように並べ替えたいのか（昇順、降順、文字列、数値…）事前に知ることはできないはずだ。では、どうして事前に設計することができたのだろうか?」という疑問です。

つまり、

- どのような規則に基づいて並べ替えるのか？（降順、昇順）
- 配列の要素には何が含まれているのか？（文字列、数値、オブジェクト）

といったことはコンテンツを作る人が決めることであって、JavaScriptを設計する人は知ることができないはずなのです。それでも、sortメソッドが用意されているのです。矛盾しているようには思いませんか? その答えはもうお気づきですね。関数オブジェクトの出番です。

Arrayのsortに渡す関数オブジェクトは何でもよいというわけではありません。以下のルールを満たす関数オブジェクトを渡す必要があります。

- 引数を2個(aとb)を受け取る
- a→bの順に並べる場合、0より小さい値を返す
- b→aの順に並べる場合、0より大きい値を返す

というルールを満たす必要があります。

Arrayクラスのsortメソッドを実行すると、配列に含まれる要素の数に応じて、自分の作成した関数オブジェクトが繰り返し呼び出されます。sortメソッドはその結果をもとに並べ替えを行ってくれます。プログラマは「ふたつの要素を引数として受け取って、どちらが前に来るか指示するだけの関数オブジェクトを用意するだけ」でよいのです。自分でソートのアルゴリズムを実装するよりもずっと楽なはずです。

説明を読むよりも、実行してみるほうがわかりやすいでしょう。以下の例をご覧ください。

```
// 例1
var data = [1, 8, 0, 6, 3, 4, 9, 2];
data.sort(function (a, b) {return a - b;})
data;   // [0, 1, 2, 3, 4, 6, 8, 9]

// 例2
data.sort(function (a, b) { return b - a; })
data;   // [9, 8, 6, 4, 3, 2, 1, 0]

// 例3
function mysort(a, b) {
    return a - b;
}
data.sort(mysort);
data;   // [0, 1, 2, 3, 4, 6, 8, 9]
```

例1は昇順の場合です。aがbよりも小さい場合、aはbよりも前に並んでほしいのでマイナスを返す必要があります。そこで、「a−b」の結果を返しています。

例2は降順の場合です。逆の順番で並べたいので「b−a」を返しています。

例3は例1の別の書き方です。例1のように無名関数を使ってもよいですし、例3のように関数名を明示的に指定することも可能です。

ほかにも例を見てみましょう。文字列の長さ順に並べる例です。

```
var data = ["hi", "hello", "say", "yes!"];
data.sort(function (a, b) { return a.length - b.length; })
data;   // hi, say, yes!, hello
```

配列の中身が独自オブジェクトの場合です。比較関数の中では、オブジェクトのageプロパティの値を比較しています。これにより、年齢が若い順にオブジェクトがソートされます。

```
var data = [
    { name: "Joe", age: 34 },
    { name: "Sam", age: 29 },
    { name: "Todd", age: 45 },
    { name: "Bill", age: 18 },
]
data.sort(function (a, b) { return a.age - b.age; })
data;   // Bill→Sam→Joe→Todd
```

このように、Arrayオブジェクトは、あなたが用意した関数オブジェクトの結果を参考に要素を並べ替えてくれるのです。関数オブジェクトの便利さを感じていただけたでしょうか。

3-11-4 | イベントハンドラも関数オブジェクト

以下は押されたキーの値を画面に表示するコードです。

```
window.onkeydown = function (e) {
    document.getElementById("result").textContent = "keydown:" + e.keyCode;
}
```

このコードは以下のように書くこともできます。

```
function mykeydown(e) {
    document.getElementById("result").textContent = "keydown:" + e.keyCode;
}
window.onkeydown = mykeydown;
```

window.onkeydownとあるのは、windowオブジェクトのonkeydownプロパティということです。この

onkeydownはイベント発生時の処理を記述するイベントハンドラを登録するためのプロパティです。ここに関数オブジェクトを代入しておくと、実際にキーが押下されたときに関数が実行されるのです。つまり、イベントハンドラとは関数オブジェクトにほかならないのです。

マウスが押された、マウスが動いた、キーが押下された、といった事象はブラウザが検出します。しかしながら、それらの事象がおきたときに何をするかはコンテンツに依存します。ブラウザを作る人はイベントが発生したときにコンテンツで何をすべきか知る由もありません。

そこで関数オブジェクトの出番となるのです。ブラウザは、事象が発生したときに、その事象に応じたイベントハンドラを呼び出すことだけ知っていればよいのです。コンテンツ作成者が「事象が発生したときに何をすべきか」を、関数オブジェクトとしてイベントハンドラに設定するのです。このような柔軟な仕組みが簡単に実現できるのも関数オブジェクトの恩恵といってよいでしょう。

ちなみに、ブラウザがページをロードしたときに呼び出すbody要素のonloadメソッド、setIntervalやsetTimeoutといったタイマー関数の引数に指定するのも関数オブジェクトです。このようにJavaScriptではさまざまな場面で関数オブジェクトの恩恵にあずかっているのです。

3-11-5 | 本章のサンプル

本章で学んだJavaScriptの知識をもとに、ゲームに取り掛かるまえの準備運動としていくつか例をご紹介しましょう。

▶インタラクティブレシピ

スライダバーで人数を変えると、それに応じて材料の量が変わります。text-shadow、box-shadow、border-radiusといったCSSスタイルを使っているところに注目してください。

インタラクティブレシピ

SAMPLE recipe.html

```
<!DOCTYPE html>
<html>
    <head>
        <style>
            h2 {
                text-align: center;
            }
            h3 {
                color: green;
                text-shadow: 2px 2px 5px;
            }
            p {
                font-size: 20px;
            }
            td {
                padding: 10px;
                background-color: #f0f0f0;
                box-shadow: rgba(0,0,0,0.4) 10px 10px 20px;
                border-radius: 10px;
            }
        </style>
        <script>
            window.onload = function () {
                document.getElementById("setNum").onchange = setNum;
            }

            function setNum(e) {
                var num = parseInt(e.target.value);
                var v0 = num, v1 = 0.5 * num, v2 = 100 * num;
                document.getElementById("i0").textContent = v0;
                document.getElementById("i1").textContent = v1;
                document.getElementById("i2").textContent = v2;
                document.getElementById("num").textContent = num
            }
        </script>
    </head>
    <body>
        <h2>ジャーマンポテト</h2>
        <p>
            <input id="setNum" type="range" min="1" max="5" value="1"/>
            <span id="num">1</span>人前
        </p>
        <table>
            <tr>
```

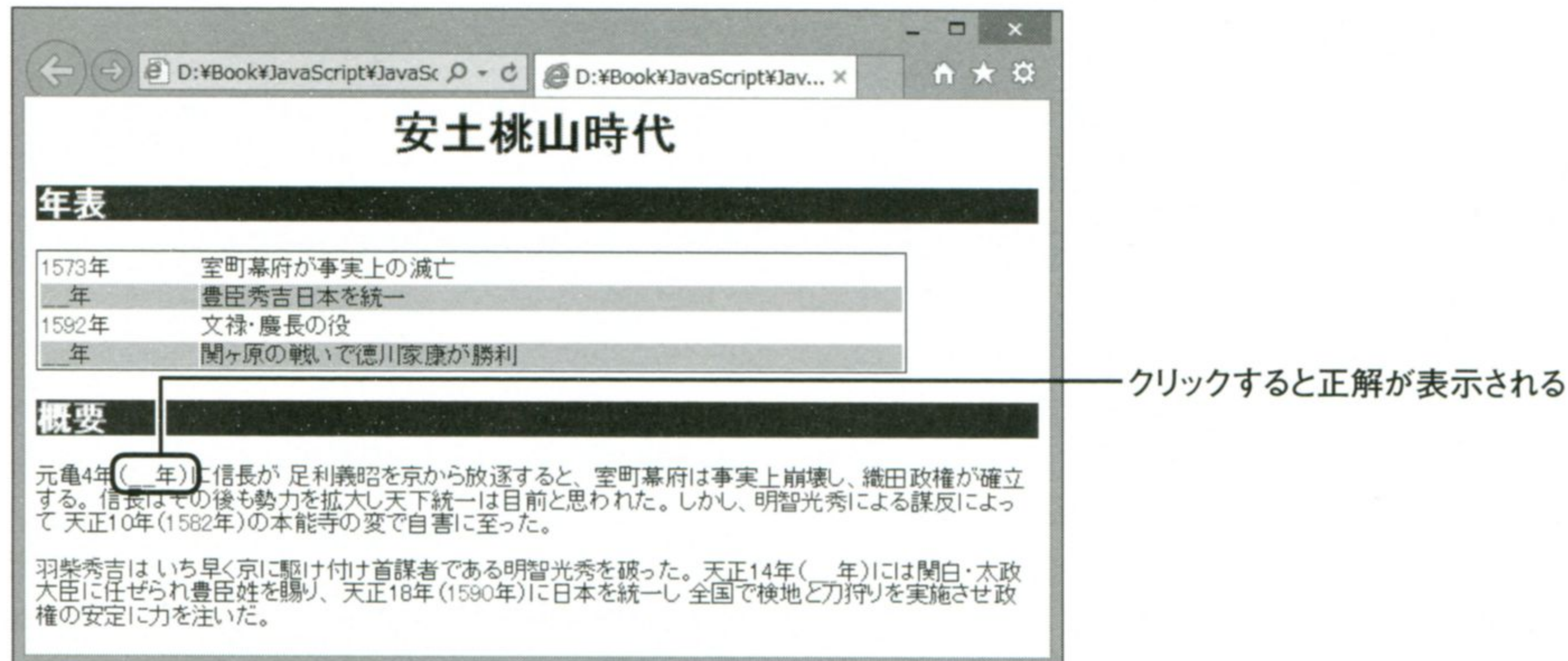
```
                <td rowspan="2">
                    <img src="germanpotate.jpg" />
                </td>
                <td>
                    <h3>材料</h3>
                    <ul>
                        <li>ジャガイモ　<span id="i0">１</span>個</li>
                        <li>玉ねぎ　<span id="i1">0.5</span>個</li>
                        <li>ベーコン　<span id="i2">100</span>g</li>
                        <li>パセリ　少々</li>
                    </ul>
                </td>
            </tr>
            <tr>
                <td>
                    <h3>作り方</h3>
                    <ol>
                        <li>ジャガイモと玉ねぎを薄切りにします</li>
                        <li>ベーコン、ジャガイモ、玉ねぎの順で炒めます</li>
                        <li>塩コショウで味を調えます</li>
                        <li>盛り付けてパセリを振りかけて出来上がり！</li>
                    </ol>
                </td>
            </tr>
        </table>

    </body>
</html>
```

▶暗記アプリ

赤色の下線部分をクリックすると正解が表示されます。JavaScriptの部分は非常にシンプルなのですぐにわかると思います。以下の例は歴史年表ですが、英単語や元素記号など自分専用の暗記用ページを作って活用してください。

「1573」で囲んだ部分が暗記対象部分となります。「var years = document.querySelectorAll("span.year")」で、その部分をすべて選択し、文字列を下線に置き換え、イベントハンドラを登録していることに注目してください。

暗記アプリ

クリックすると正解が表示される

SAMPLE history.html

```
<html>
<head>
    <style>
        h1 { text-align: center; }
        h2 {
            color: white;
            background-color: blue;
        }
        #history {
            border: 1px solid blue;
            width:600px;
        }
        span.year {
            color: red;
        }
        span.name {
            color: blue;
        }
        tr:nth-child(2n) {
            background-color: lightblue;
        }
```

```
    </style>
    <script>
        window.onload = init;
        function init() {
            var years = document.querySelectorAll("span.year");
            for(var i = 0 ; i < years.length ; i++){
                var y = years[i];
                y.answer = y.textContent;
                y.textContent = "____";
                y.onclick = function (e) {
                    e.target.textContent = e.target.answer;
                }
            }
        }
    </script>
</head>
<body>
    <h1>安土桃山時代</h1>
    <h2>年表</h2>
    <table id="history">
        <tr><td><span class="year">1573</span>年</td><td>室町幕府が事実上の滅亡</td></tr>
        <tr><td><span class="year">1590</span>年</td><td>豊臣秀吉日本を統一</td></tr>
        <tr><td><span class="year">1592</span>年</td><td>文禄・慶長の役</td></tr>
        <tr><td><span class="year">1600</span>年</td><td>関ヶ原の戦いで徳川家康が勝利</td></
tr>
    </table>
    <h2>概要</h2>
    <p>
        元亀4年（<span class="year">1573</span>年）に信長が
        <span class="name">足利義昭</span>を京から放逐すると、
        室町幕府は事実上崩壊し、織田政権が確立する。
        信長はその後も勢力を拡大し天下統一は目前と思われた。
        しかし、<span class="name">明智光秀</span>による謀反によって
        天正10年（<span class="year">1582</span>年）の本能寺の変で自害に至った。
    </p>
    <p>
        <span class="name">羽柴秀吉</span>は
        いち早く京に駆け付け首謀者である<span class="name">明智光秀</span>を破った。
        天正14年（<span class="year">1586</span>年）には関白・太政大臣に任ぜられ豊臣姓を賜り、
        天正18年（<span class="year">1590</span>年）に日本を統一し
        全国で検地と刀狩りを実施させ政権の安定に力を注いだ。
    </p>
</body>
</html>
```

▶動くカレンダー

カレンダーの写真部分をクリックすると動画が再生されます。ビデオは「<video src="video.MP4" onclick="play()"/>」のようにvideo要素を使います。クリックするとplay()が呼ばれ、そこでdocument.getElementById("video").play()が実行され、再生が始まります。この例はCSSのセレクタに多少工夫をしています。

CSSのセレクタ

セレクタ	説明
tr:nth-child(2)	同じ親を持つ中で２番目のtr要素（曜日の文字を斜体に）
td:first-child	同じ親を持つ中で最初のtd要素（日曜日の列を赤色に）

カレンダー

クリックすると動画が再生される

SAMPLE calendar.html

```
<!DOCTYPE html>
<html>
    <head>
        <style>
            video {
                width:600px;
                box-shadow: 10px 10px 10px rgba(0,0,0,0.4);
            }
            h2 {
                color: #0094ff;
                text-align: center;
            }
            td {
                font-size:28px;
                text-align: center;
```

```
            border: 1px solid #cccccc;
            border-radius: 5px;
        }
        .red {
            color:red;
        }
        tr:nth-child(2) {
            font-weight: bold;
            font-style: italic;
        }
        td:first-child {
            color: red;
        }
        table {
            margin: 20px;
        }
    </style>
    <script>
        function play() {
            document.getElementById("video").play();
        }
    </script>
</head>
<body>
    <h2>2020年7月カレンダー</h2>
    <table>
        <tr>
            <td colspan="7">
                <video src="video.MP4" onclick="play()"/>
            </td>
        </tr>
        <tr>
            <td>日</td>
            <td>月</td>
            <td>火</td>
            <td>水</td>
            <td>木</td>
            <td>金</td>
            <td>土</td>
        </tr>
        <tr>
            <td></td>
            <td></td>
            <td></td>
            <td>1</td>
```

```
            <td>2</td>
            <td>3</td>
            <td>4</td>
        </tr>
        <tr>
            <td>5</td>
            <td>6</td>
            <td>7</td>
            <td>8</td>
            <td>9</td>
            <td>10</td>
            <td>11</td>
        </tr>
        <tr>
            <td>12</td>
            <td>13</td>
            <td>14</td>
            <td>15</td>
            <td>16</td>
            <td>17</td>
            <td>18</td>
        </tr>
        <tr>
            <td>19</td>
            <td class="red">20</td>
            <td>21</td>
            <td>22</td>
            <td>23</td>
            <td>24</td>
            <td>25</td>
        </tr>
        <tr>
            <td>26</td>
            <td>27</td>
            <td>28</td>
            <td>29</td>
            <td>30</td>
            <td>31</td>
            <td></td>
        </tr>
    </table>
  </body>
</html>
```

1章からここまで、HTML、CSS、JavaScriptと説明してきました。量が多いと思われたかもしれません。肝となる概念についてはひととおりカバーできたと思いますが、要素、CSSスタイル、組み込みオブジェクトのメソッド・プロパティなど説明できたのはほんの一部です。

しかし、基本ができていれば自分で調べて引き出しを増やしていくことができるはずです。引き出しを増やすには、さまざまなソースに接して、自分で試してみて、というプロセスが欠かせません。これ以降もたくさんのサンプルゲームが出てきますが、ぜひ自分で入力して試行錯誤してください。

演習　これまでの知識をもとに自由にWebページをつくってみよう

HTML、CSS、JavaScriptの基本はカバーしました。自分自身のアイディアを活かして楽しいページをつくってみましょう。クリックすると正解が表示される元素周期表、正解するまで問題が繰り返し表示される英単語帳、いつも乗る電車の時刻表、等々。これまで出てきた例をベースに自分なりに修正してみてもよいでしょう。

Chapter 4

Canvasの基本

CanvasはHTML5で新しく導入された要素です。Webページ上に、線、矩形、円、ポリゴン、画像などいろいろなものを描画することができます。Webページの可能性は飛躍的に広がり、特にゲームでは欠かせない要素となりました。本章では、このCanvasの使い方について説明していきます。

4-1 canvas要素で図形を描く

Canvasはで追加された要素で、線や矩形、円、画像などを描画できます。さらに、座標系を変換したり、一部分を切り抜いたりと、さまざまな機能が利用できます。

4-1-1 描画の手順

ある人が「あっ、くも!」と叫びました。おそらく「蜘蛛」を連想した人が多いと思います。「見て、あのくも!」だったら、「雲」を思い浮かるのではないでしょうか? 同じ「くも」でも状況によって連想する内容が違ってきます。このように物事を判断するために必要な情報をコンテキスト（文脈、前後関係、背景）といいます。

Canvasとまったく関係ないと思うかもしれませんが、Canvasを習得する鍵はコンテキストにあります。同じように筆を動かしたとしても、コンテキストが違えば描かれる結果が異なってきます。筆を持つ手を同じように動かしても（同じような処理を行っても）、そのときの絵の具や筆の状態によって描画される内容（色や太さ）が変わるのと同じです。

Canvasに描画する際は、

① HTMLでcanvas要素を定義
② JavaScriptでcanvas要素への参照を取得
③ canvas要素の参照からコンテキストを取得
④ コンテキストに色や線の太さなどを設定
⑤ コンテキストに対して線や矩形などの描画を行う

という手順になります。では、さっそく例を見てみましょう。

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            var ctx;
            function init() {
                var canvas = document.getElementById("canvas"); ←1
                ctx = canvas.getContext("2d"); ←2

                // コンテキストの設定
                ctx.strokeStyle = "#FF0000";
                ctx.fillStyle = "#00FF00";
                ctx.lineWidth = 10;                                ←3
                ctx.lineCap = "round";
                ctx.shadowColor = "#000000";
                ctx.shadowBlur = 20;

                // 線を引く
                ctx.beginPath();
                ctx.moveTo(100, 100);
                ctx.lineTo(180, 250);
                ctx.stroke();                                      ←4

                // 矩形を塗りつぶす
                ctx.fillRect(300, 100, 100, 150);

                // 矩形を描く
                ctx.strokeRect(500, 100, 100, 150);
            }
        </script>
    </head>
    <body onload="init()">
        <canvas id="canvas" width="700" height="400"></canvas> ←5
    </body>
</html>
```

HTMLではcanvas要素を定義します5。

```
<canvas id="canvas" width="700" height="400"></canvas>
```

ここで注意したいのは幅をwidth属性で、高さをheight属性で指定することです。この指定が正しくなされていないと意図した大きさで描画されないことがあります。

JavaScriptでは、1の

```
var canvas = document.getElementById("canvas");
```

でcanvasへの参照を取得し、次に2の

```
ctx = canvas.getContext("2d");
```

でコンテキストを取得します。このctxが絵筆などの情報を格納するオブジェクトとなります。

多くのプログラムでは、コンテキストをctxやcontextといった広域変数に格納しているようです。引数に「2d」とありますが、現在は「2d」しか指定できません。おそらく将来的には3次元の仕様が策定されることを見越してこのような仕様になっているのでしょう。

コンテキストを取得したらプロパティ（絵筆の属性）を設定します3。主なプロパティは以下のとおりです。

図形描画の主なプロパティ

プロパティ	内容
ctx.strokeStyle	線や輪郭の色
ctx.fillStyle	塗りつぶしの色
ctx.lineWidth	線の幅
ctx.lineCap	線の終端の形状で、butt、round、squareの値が使用可能
ctx.shadowColor	現在の影の色
ctx.shadowBlur	影に適用するぼかす範囲

コンテキストにプロパティを設定したのち、線、矩形の塗りつぶし、矩形と描画しています4。

描画する際に筆を動かすように、Canvasでは必ず「パス」と呼ばれる軌跡を設定します。そのパスを初期化するのがbeginPath()です。moveTo()で筆を下ろし、lineTo()で筆を動かします。ただ、これだけでは画面上になにも表示されません。stroke()で初めて線が描画されます。ちなみに、fill()ではパスで囲まれた範囲が塗りつぶされます。fillRect()とstrokeRect()は名前から予想できると思いますが、塗りつぶし、矩形の描画です。座標系はマウスイベントと同じく、Canvasの左上を原点(0,0)、右方向にx軸、下方向にy軸となります。

線、塗りつぶし、矩形の描画

(0,0)
X軸方向
Y軸方向

（各図形共通）
```
ctx.shadowColor = "#000000";
ctx.shadowBlur = 20;
```

(100,100)
(180,250)
(300,100)
100
150
(500,100)
100
150

```
ctx.strokeStyle = "#FF0000";
ctx.lineWidth = 10;
ctx.lineCap = "round";

// 線を引く
ctx.beginPath();
ctx.moveTo(100, 100);
ctx.lineTo(180, 250);
ctx.stroke();
```

```
ctx.fillStyle = "#00FF00";

// 矩形を塗りつぶす
ctx.fillRect(300, 100, 100, 150);
```

```
ctx.strokeStyle = "#FF0000";
ctx.lineWidth = 10;

// 矩形を描く
ctx.strokeRect(500, 100, 100, 150);
```

演習　Canvasで図形を描いてみよう

Canvasを使って、いろいろな線、矩形を描画してください。

4-2 さまざまな図形の描画

前節で見たように、Canvasでは、コンテキストにプロパティを設定し、描画用の命令を実行するという手順で画面上に図形を描きます。ここでは、図形の種類別に設定できる属性や描き方の概要を見ていきます。

4-2-1 直線・多角形

直線や多角形を描画するために必要な主なメソッド、プロパティは以下のとおりです。

直線、多角形描画の主なメソッド、プロパティ

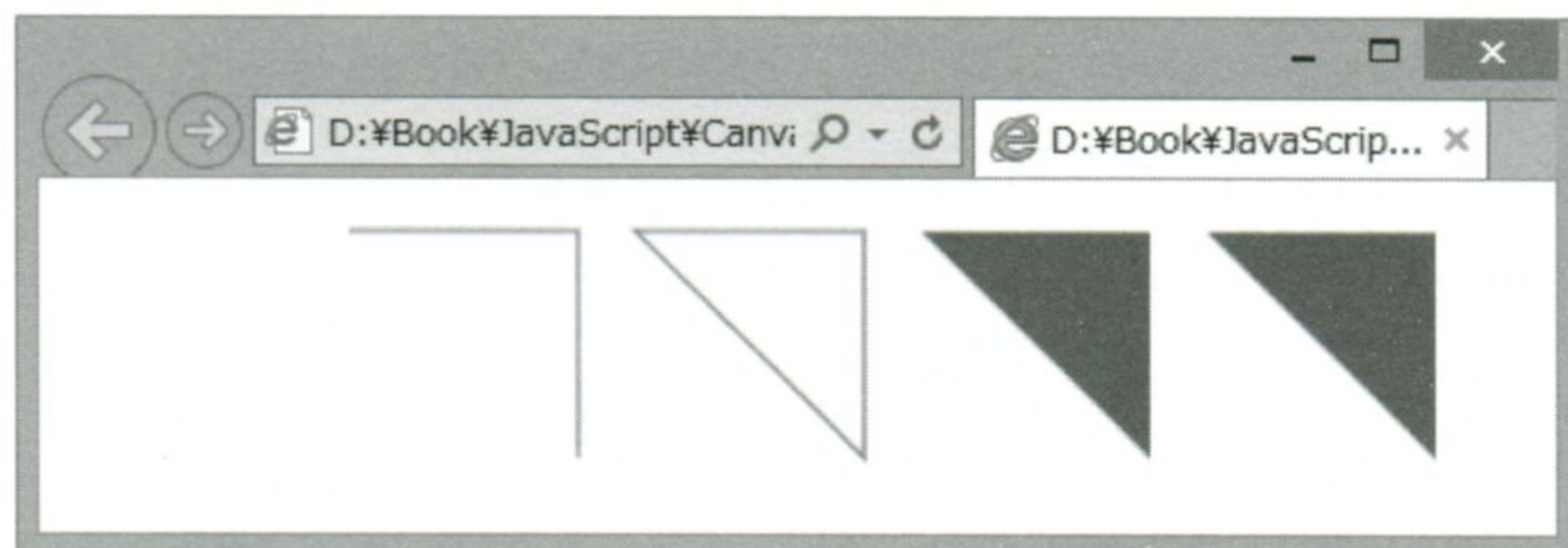
メソッド、プロパティ	説明
beginPath()	パスをクリアする
moveTo(x, y)	(x, y)座標に筆を下ろす
lineTo(x, y)	(x, y)座標に筆を動かす
closePath()	パスを閉じる（筆を始点まで動かす）
stroke()	筆が動いた軌跡を線として描く
fill()	筆が動いた軌跡の範囲を塗りつぶす
strokeStyle	線を描画する際の色を指定する
fillStyle	塗りつぶしの色を指定する

パスを閉じた場合と閉じない場合、それぞれについて、塗りつぶし、軌跡で描いた4パターンのサンプルを以下に示します。

SAMPLE canvas-line.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            var ctx;
            function init() {
                var canvas = document.getElementById("canvas");
                ctx = canvas.getContext("2d");

                ctx.strokeStyle = "#FF0000";
                ctx.fillStyle = "#00FF00";
```

```
                drawTriangle(100, 10, false, false);
                drawTriangle(200, 10, true, false);
                drawTriangle(300, 10, false, true);
                drawTriangle(400, 10, true, true);
            }

            function drawTriangle(x, y, isClose, isFill) {
                ctx.beginPath();
                ctx.moveTo(x, y);
                ctx.lineTo(x + 80, y);
                ctx.lineTo(x + 80, y + 80);
                if (isClose) {
                    ctx.closePath();
                }
                if (isFill) {
                    ctx.fill();
                } else {
                    ctx.stroke();
                }
            }
        </script>
    </head>
    <body onload="init()">
        <canvas id="canvas" width="500" height="100"></canvas>
    </body>
</html>
```

ブラウザ表示例

パスを初期化してない、パスを閉じていないために意図した描画にならないことはよくあるミスです。線を描画するときは必ずbeginPath()を実行する習慣をつけるとよいでしょう。

演習 **マウスの軌跡を線で描いてみよう**

マウスを押下した点から離した点までの線分を描画するページを作成してください。次に、マウスの軌跡を描画するページを作成してみましょう。500x500の領域を設定して、その中に描画するようにします。線分を描画する場合は、canvasにonmousedown、onmouseupのイベントハンドラを登録し、マウスが押下された座標、マウスが離された座標を取得します。軌跡を描画する場合は、さらにonmousemoveのイベントハンドラを使用し、マウスが動いた座標をすべて配列に格納します。

SAMPLE canvas-line-mouse0.html、canvas-line-mouse1.html

4-2-2 | 矩形

矩形を描画するために必要なメソッドは以下のとおりです。矩形の描画はパスに影響を与えないのでbeginPath()やclosePath()を実行する必要はありません。

矩形描画のメソッド

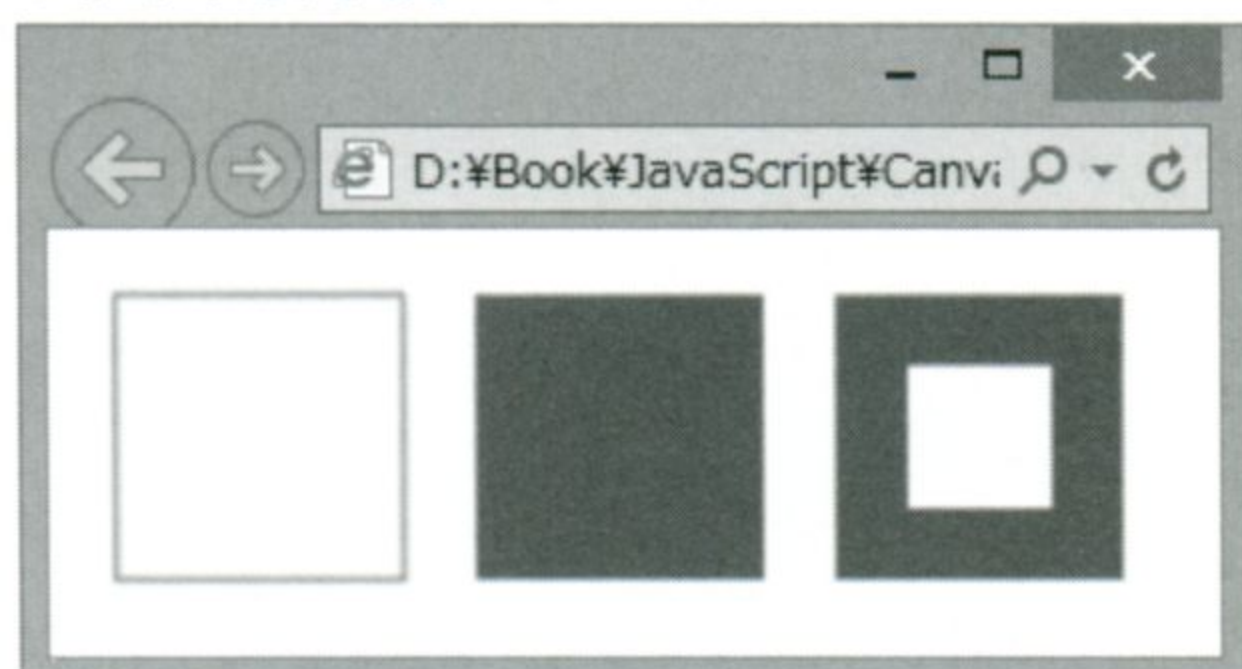
メソッド	説明
strokeRect(x, y, w, h)	(x,y)を左上隅として、幅w、高さhの矩形の輪郭を描く
fillRect(x, y, w, h)	(x,y)を左上隅として、幅w、高さhの矩形を塗りつぶす
clearRect(x, y, w, h)	(x,y)を左上隅として、幅w、高さhの矩形をクリアする

SAMPLE canvas-rect.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            var ctx;
            function init() {
                var canvas = document.getElementById("canvas");
                ctx = canvas.getContext("2d");

                ctx.strokeStyle = "#FF0000";
                ctx.strokeRect(10, 10, 80, 80);

                ctx.fillStyle = "#00FF00";
                ctx.fillRect(110, 10, 80, 80);

                ctx.fillRect(210, 10, 80, 80);
                ctx.clearRect(230, 30, 40, 40);
            }
        </script>
    </head>
```

```
    <body onload="init()">
        <canvas id="canvas" width="300" height="100"></canvas>
    </body>
</html>
```

ブラウザ表示例

D:¥Book¥JavaScript¥Canvi

4-2-3 | 円、円弧

円を描画するために必要なメソッドは以下のとおりです。

```
arc(x, y, radius, startAngle, endAngle, anticlockwise)
```

(x,y)を中心とし、半径radiusで、startAngleからendAngleまでの弧のパスを描きます。startAngleは円弧の描画開始角度、endAngleは描画終了角度になります。

arcは単にパスを描くだけのメソッドなので、実際に描画するためには、stroke()かfill()を実行する必要があることに注意してください。

角度は「ラジアン」という単位で指定します。これは1回転（360度）を2×π（=3.1415…×2）とする単位です。度数からラジアンへの変換は「度数×π÷180」で求められます。たとえば、60度はラジアンで表すと「60×π÷180=π÷3」となります。anticlockwiseにtrueを指定すると反時計回り、falseで時計回りとなります。

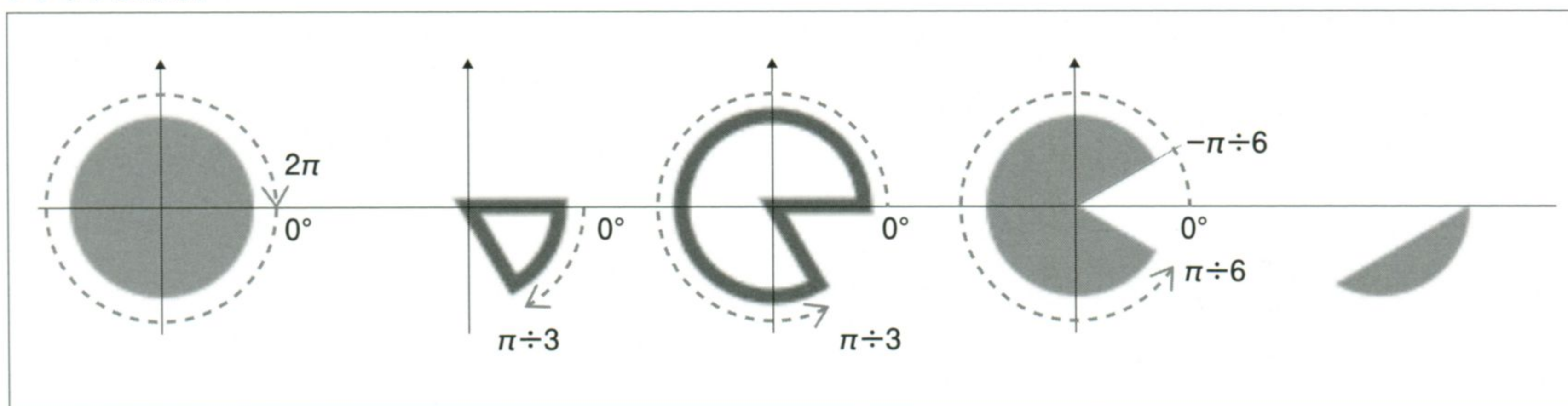
SAMPLE canvas-circle.html

```
<!DOCTYPE html>
<html>
    <head>
        <META charset="UTF-8">
        <script>
            var ctx;
            function init() {
                var canvas = document.getElementById("canvas");
```

```
            ctx = canvas.getContext("2d");
            ctx.strokeStyle = "#FF0000";
            ctx.fillStyle = "#00FFFF";
            ctx.lineWidth = 5;

            ctx.beginPath();    // 1:円
            ctx.arc(100, 50, 30, 0, 2 * Math.PI);
            ctx.closePath();
            ctx.fill();

            ctx.beginPath();    // 2:扇型（時計回り）
            ctx.moveTo(200, 50);
            ctx.arc(200, 50, 30, 0, Math.PI / 3);
            ctx.closePath();
            ctx.stroke();

            ctx.beginPath();    // 3:扇型（反時計回り）
            ctx.moveTo(300, 50);
            ctx.arc(300, 50, 30, 0, Math.PI / 3, true);
            ctx.closePath();
            ctx.stroke();

            ctx.beginPath();    // 4:扇型（反時計回り）
            ctx.moveTo(400, 50);
            ctx.arc(400, 50, 30, -1 * Math.PI / 6, Math.PI / 6, true);
            ctx.closePath();
            ctx.fill();

            ctx.beginPath();    // 5:円弧のパスのみ
            ctx.arc(500, 50, 30, 0, 2 * Math.PI / 3);
            ctx.fill();
        }
    </script>
  </head>
  <body onload="init()">
    <canvas id="canvas" width="600" height="400"></canvas>
  </body>
</html>
```

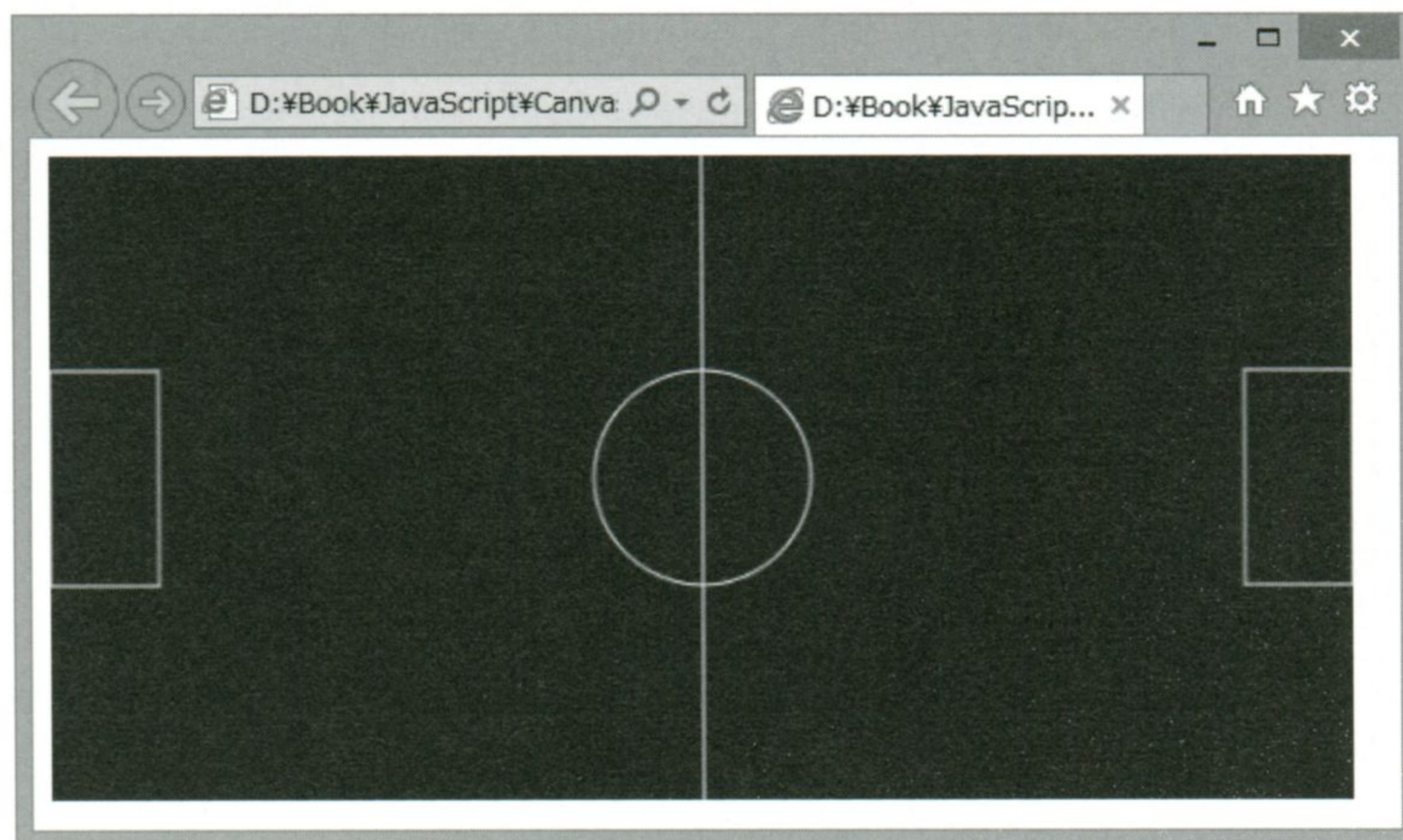
上記のサンプルにおけるパラメータと実際の描画の様子を次に示します。

ブラウザ表示例

2π
0°
0°
π÷3
0°
π÷3
−π÷6
0°
π÷6

円の場合は、開始角に0、終了角に2×πを指定します。円弧の場合は開始角と終了角を指定しますが、円弧を描画する方向によって結果が大きく異なることに注意してください。また、arcは単に円弧のパスを描くだけです。

演習 サッカー場をつくってみよう

Canvasを使って、図のようなサッカー場を描画してください。

D:¥Book¥JavaScript¥Canva
D:¥Book¥JavaScrip...

SAMPLE canvas-soccer.html

(4-2-4 | 文字)

文字を描画するために必要なメソッドは以下のとおりです。

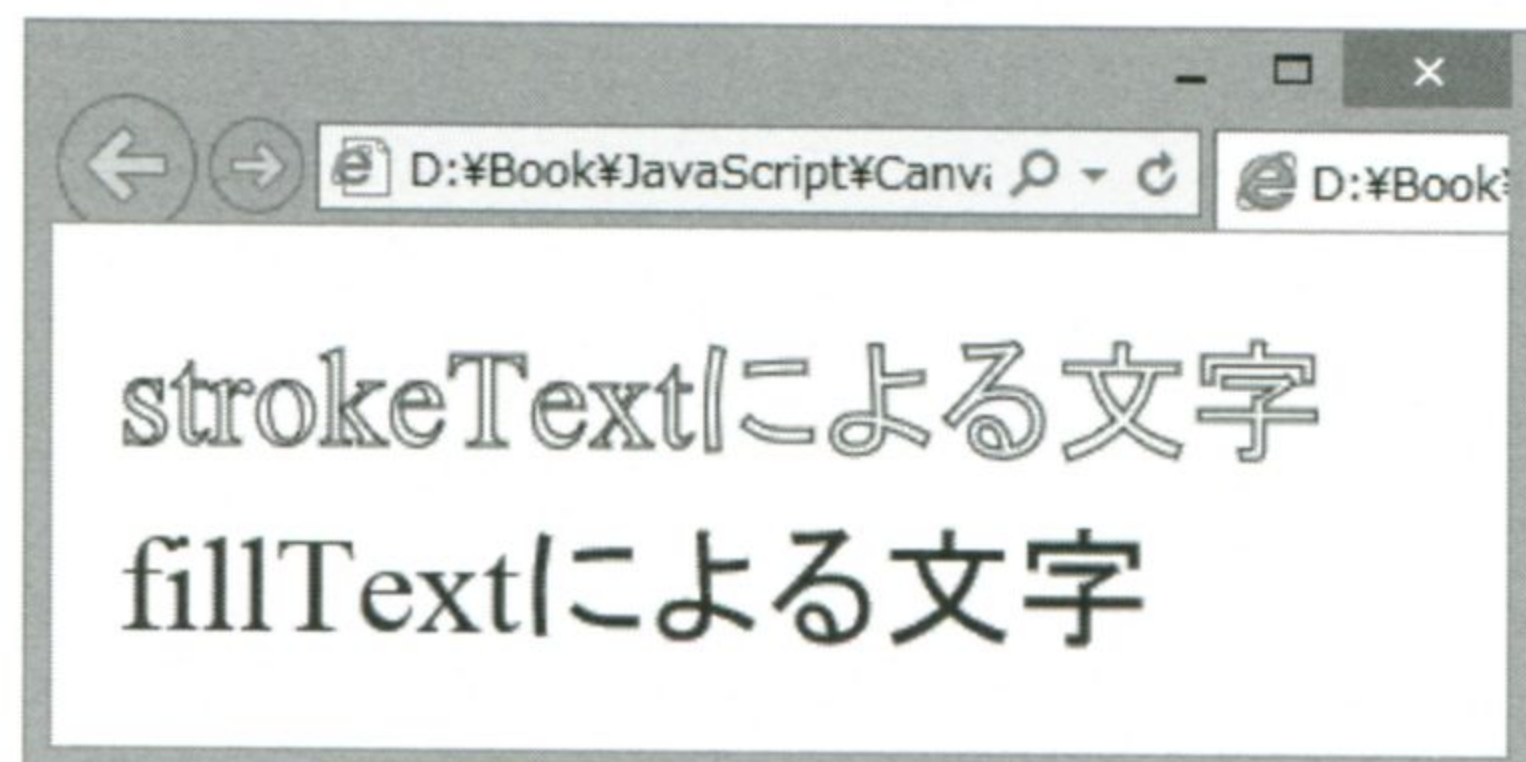
文字描画のメソッド

メソッド	説明
strokeText(text, x, y)	x,y座標を起点としてtextの輪郭を描く
fillText(text, x, y)	x,y座標を起点としてtextを塗りつぶす

SAMPLE canvas-strings.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <script>
            var ctx;
            function init() {
                var canvas = document.getElementById("canvas");
                ctx = canvas.getContext("2d");
                ctx.font = "36px 'Times New Roman'";

                ctx.strokeStyle = "blue";
                ctx.strokeText("strokeTextによる文字", 10, 50);

                ctx.fillStyle = "green";
                ctx.fillText("fillTextによる文字", 10, 100);
            }
        </script>
    </head>
    <body onload="init()">
        <canvas id="canvas" width="350" height="120"></canvas>
    </body>
</html>
```

ブラウザ表示例

D:¥Book¥JavaScript¥Canv
D:¥Book
strokeTextによる文字
fillTextによる文字

4-2-5 画像

画像を描画するためのメソッドは以下のとおりです。

画像を描画するためのメソッド

メソッド	説明
drawImage(image, x, y, w, h)	x,yの位置にw,hのサイズで画像imageを描画する。w、h を省略したときは元の画像のサイズで描画される

drawImageの最初の引数であるimageには<img>要素を指定します。ただし、HTMLページ中に<img>要素を記述すると、その部分に画像が表示されてしまいます。canvasの上に描画したいのに、HTML要素として画面上に表示されるのは意図した挙動ではありません。そこで、

```
<img id="fruit0" src="fruit0.png" style="display:none" />
```

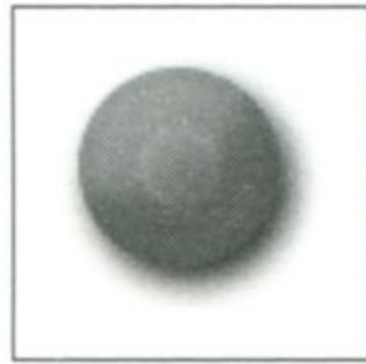
のように「style="display:none"」属性を指定して、HTMLページとしては<img>要素を非表示にします。

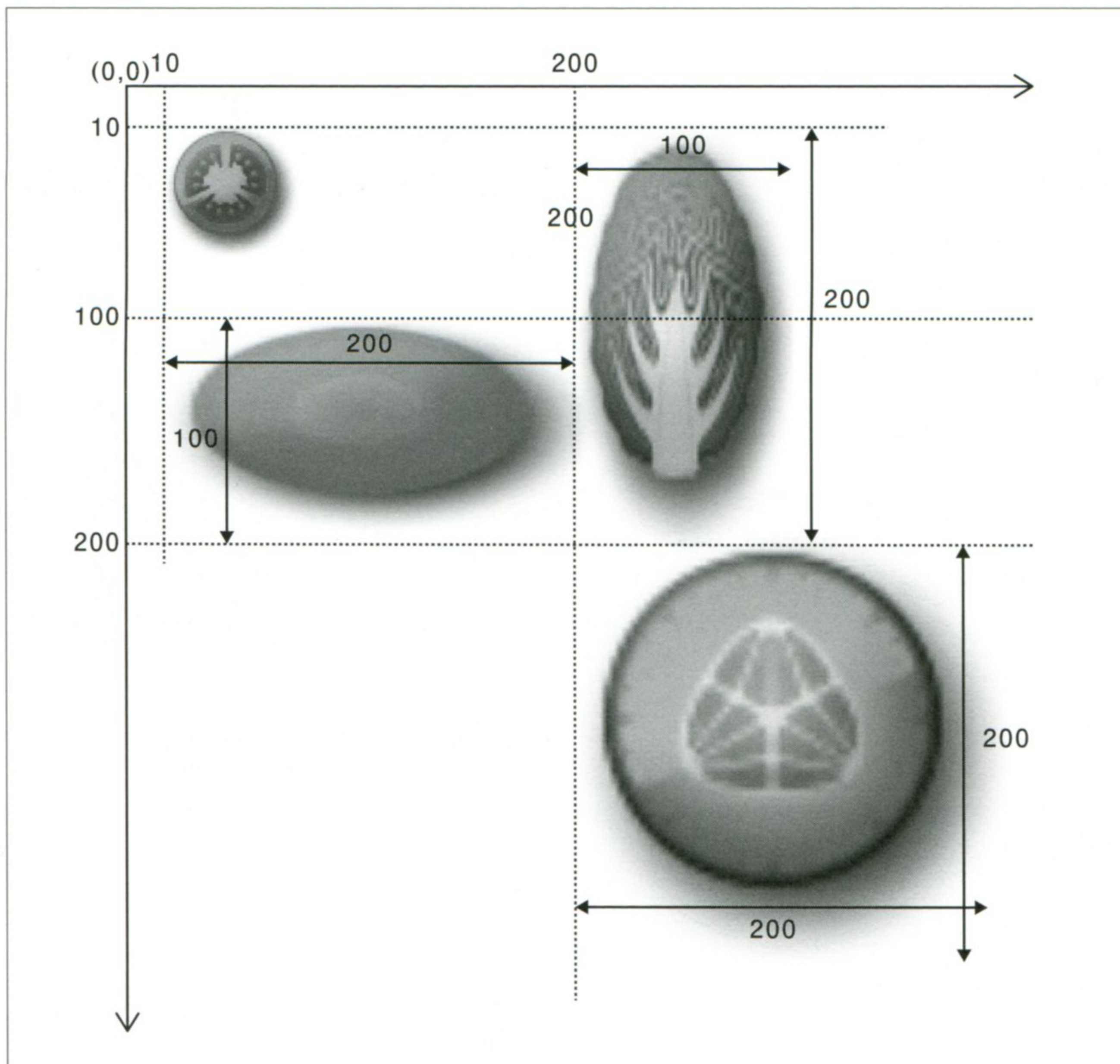
SAMPLE canvas-image.html

```
<html>
    <head>
    <meta charset="UTF-8">
    <style>
        #field {
            width: 800px; height: 800px;
        }
    </style>
    <script>
        function init() {
            var fruit0 = document.getElementById("fruit0");
            var fruit1 = document.getElementById("fruit1");
            var fruit2 = document.getElementById("fruit2");
            var fruit3 = document.getElementById("fruit3");

            var field = document.getElementById("field");
            var ctx = field.getContext("2d");

            ctx.drawImage(fruit0, 10, 10);
            ctx.drawImage(fruit1, 200, 10, 100, 200);
            ctx.drawImage(fruit2, 10, 100, 200, 100);
            ctx.drawImage(fruit3, 200, 200, 200, 200);
        }
```

```
    </script>
    </head>
    <body onload="init()">
        <canvas id="field" width="800" height="800"></canvas>
        <img id="fruit0" src="fruit0.png" style="display:none" />
        <img id="fruit1" src="fruit1.png" style="display:none" />
        <img id="fruit2" src="fruit2.png" style="display:none" />
        <img id="fruit3" src="fruit3.png" style="display:none" />
    </body>
</html>
```

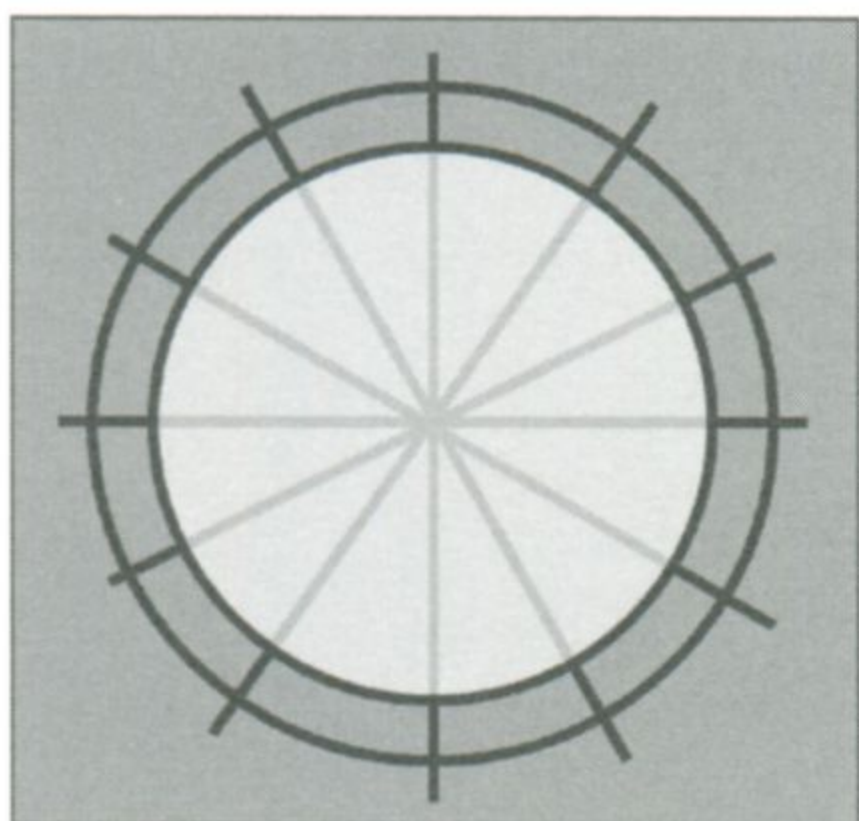
使用した画像

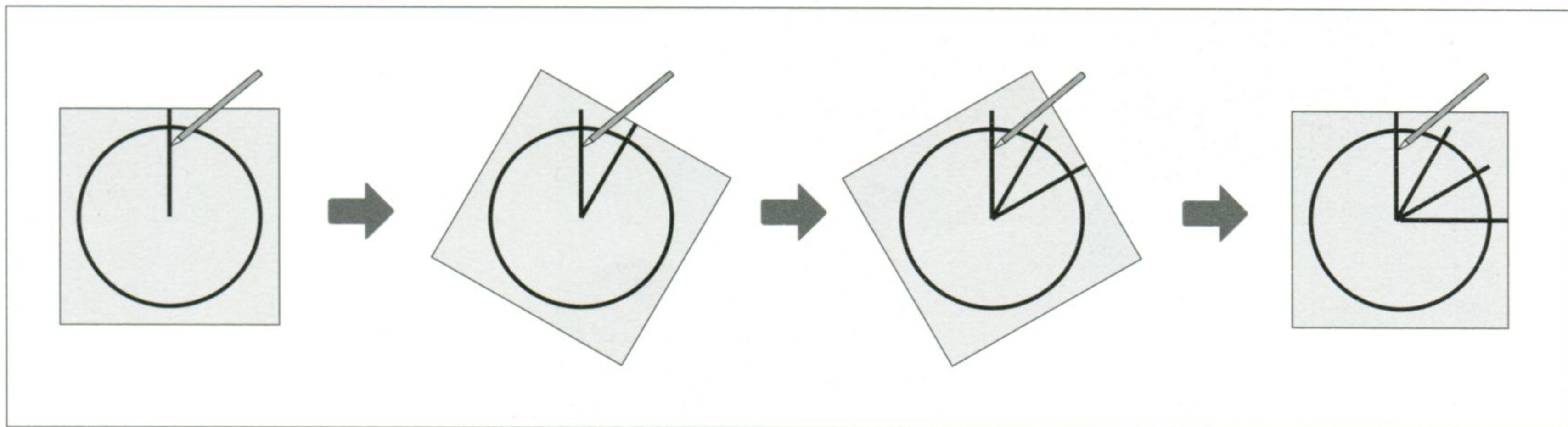
fruit0.png　fruit1.png　fruit2.png　fruit3.png

「ctx.drawImage(fruit0, 10, 10)」のようにサイズを指定しない場合は、オリジナルの画像サイズで描画されます。fruit1〜fruit3は指定したサイズで描画されていることがわかります。

画像の描画

4-3 座標系の設定

Canvasを使うと、文字や図形、画像などいろいろ描画できることがわかりました。さらに、Canvasには座標系を変換するという強力な機能が用意されています。これによって、たとえば、時計のような均等に回転する図形も座標系を回転させて簡単に描くことができます。

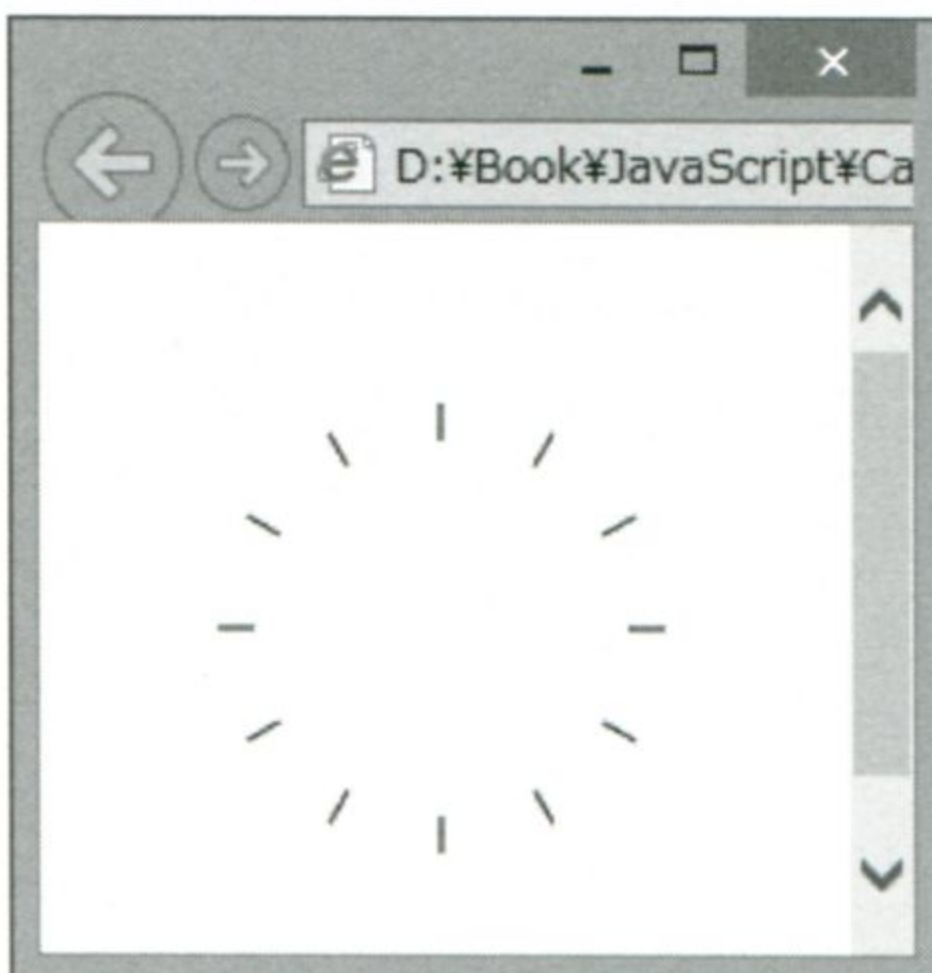
4-3-1 座標系の基礎

たとえば、次のような時計の文字盤を描くとしましょう。これは、それぞれの座標を求めて、moveToやlineToを使って線を引くことで描画できます。

しかし、実際にみなさんが手書きで紙に描いていくときは、おそらく、下図のように紙を回転させるのではないでしょうか?

紙を回転させて描画

これこそが座標系の変換にほかなりません。実際の例を見てみましょう。

SAMPLE canvas-clock0.html

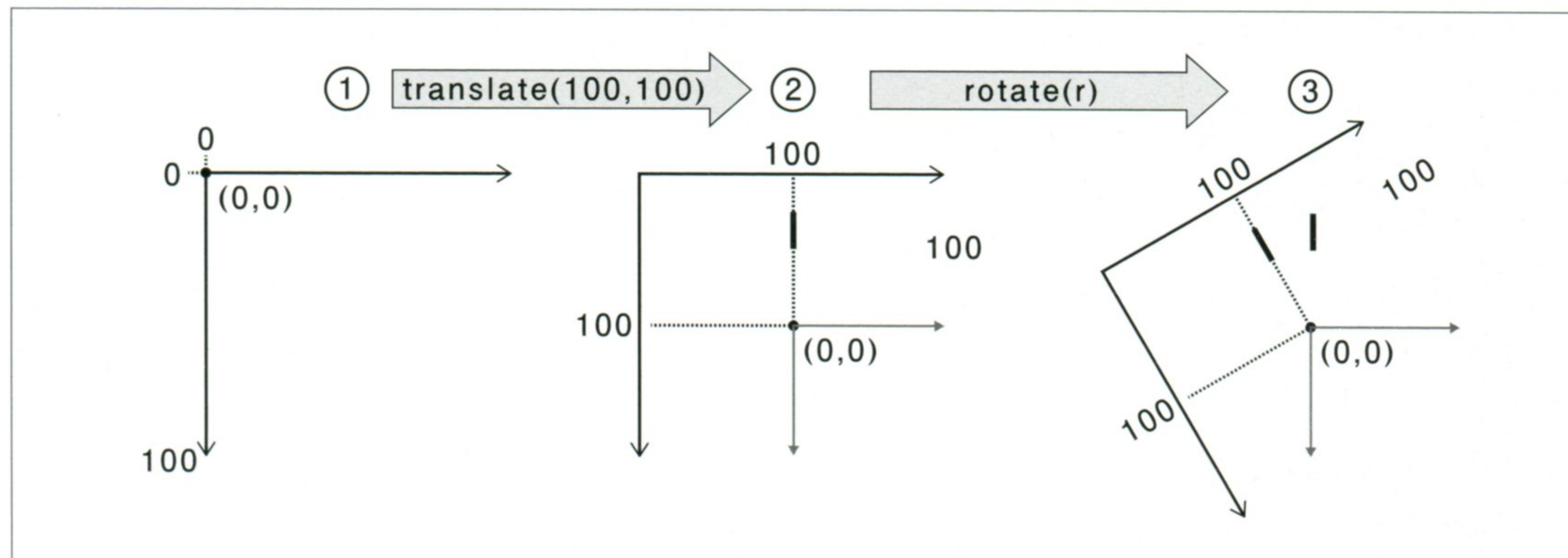
```
<!DOCTYPE html>
<html>
    <head>
        <META charset="UTF-8">
        <script>
            var ctx;
            function init() {
                var canvas = document.getElementById("canvas");
                ctx = canvas.getContext("2d");

                for (var i = 0 ; i < 12 ; i++) {
                    ctx.save(); ←1
                    var r = Math.PI / 6 * i;
                    ctx.translate(100, 100);    ←2
                    ctx.rotate(r); ←3

                    ctx.beginPath();
                    ctx.moveTo(0, -60);
                    ctx.lineTo(0, -50); ←4
                    ctx.stroke();

                    ctx.restore(); ←5
                }
            }

        </script>
    </head>
    <body onload="init()">
        <canvas id="canvas" width="200" height="200"></canvas>
    </body>
</html>
```

ブラウザ表示例

このプログラムで肝となるのは次の手順とメソッドです。

1 ctx.save() …… コンテキスト（座標系）を保存
2 ctx.translate(100, 100) …… 座標系の原点をx方向に100、y方向に100移動
3 ctx.rotate(r) …… 座標系をr回転
4 描画
5 ctx.restore() …… コンテキストを1で保存したものに復元

初期状態は下図①のように左上を原点とする座標系です。これをctx.save()で保存しておきます。

次に、「ctx.translate(100, 100)」を実行すると、(100, 100)を新たな原点とする座標系に変換されます。その状態で(0, -60)から(0, -50)に線を引くと下図②のように線が引かれます。ctx.restore()を実行するとctx.save()が実行されたときの座標系に復元されます。初回は回転角が0ですが、for文で徐々に座標系を回転させながら描画すると③のようになります。

座標系を回転させながら描画

これを繰り返すと時計の文字盤が描画されることになります。

演習 時計をつくってみよう

タイマーを使って、実際に動く時計を描画してみましょう。ヒントは以下のとおりです。

```
// 時、分、秒を求める
var now = new Date();
h = now.getHours() % 12;
m = now.getMinutes();
s = now.getSeconds();
```

```
// 長針、短針、秒針の角度（ラジアン）を求める
var radH = (Math.PI * 2) / 12 * h + (Math.PI * 2) / 12 * (m / 60);
var radM = (Math.PI * 2) / 60 * m;
var radS = (Math.PI * 2) / 60 * s;
```

07:35:20 GMT+0900 (東京 (標準時))

SAMPLE canvas-clock1.html

実践：ゲームプログラミング

Chapter 5

HTML、CSS、JavaScriptの基本を習得したら、いよいよゲームプログラミングに挑戦です！　わからないこともあるかもしれませんが、とにかく習うより慣れろ、ひたすら打ち込んで、動作を確認し、ゲームを楽しんでください。設定をいろいろ変えていくうちに、JavaScriptプログラミングのコツや書き方ががわかってくるでしょう。

HTML5
CSS
JavaScript
Canvas
Game
and
Physics engine

5-1 15 Puzzle

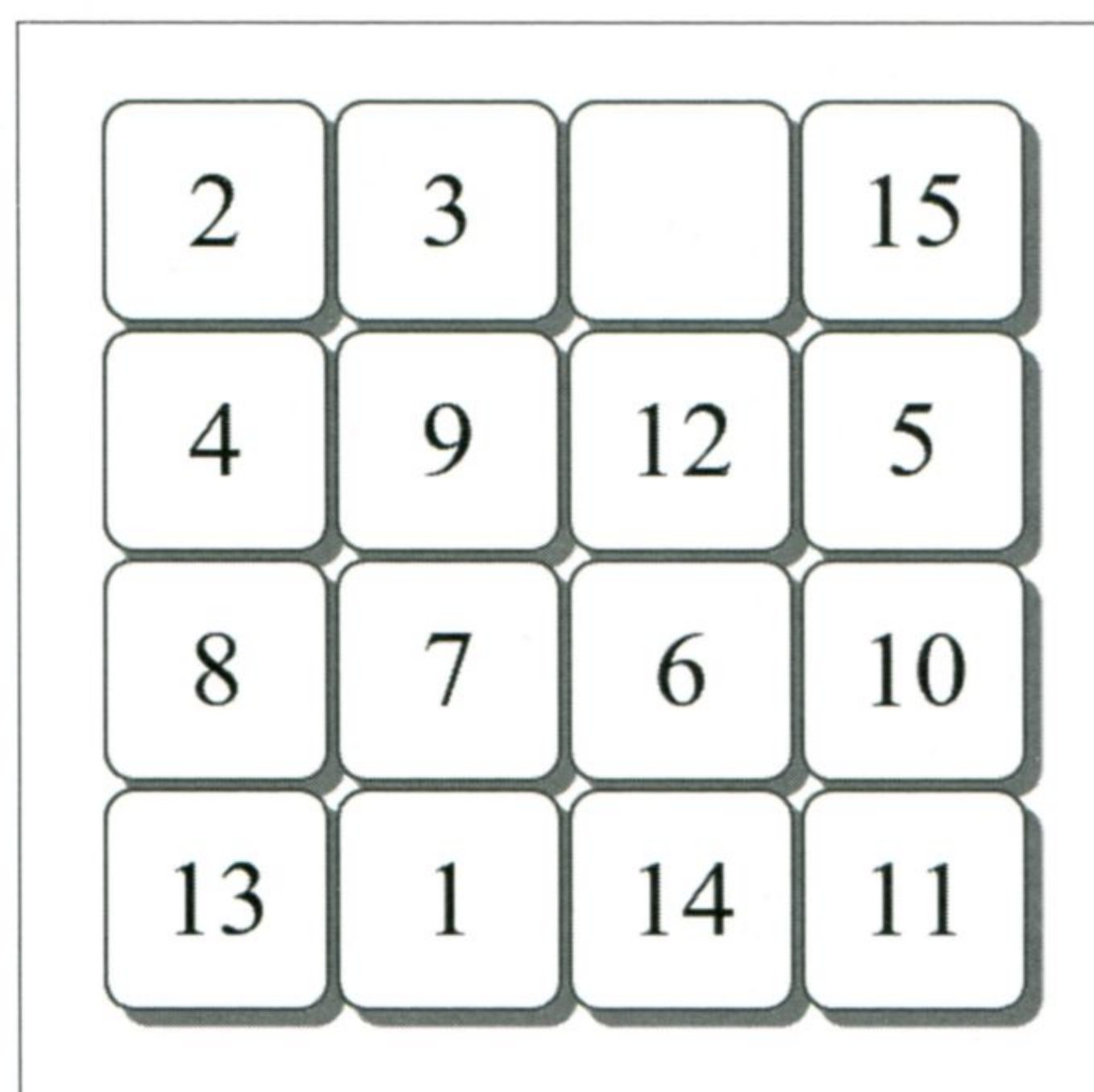
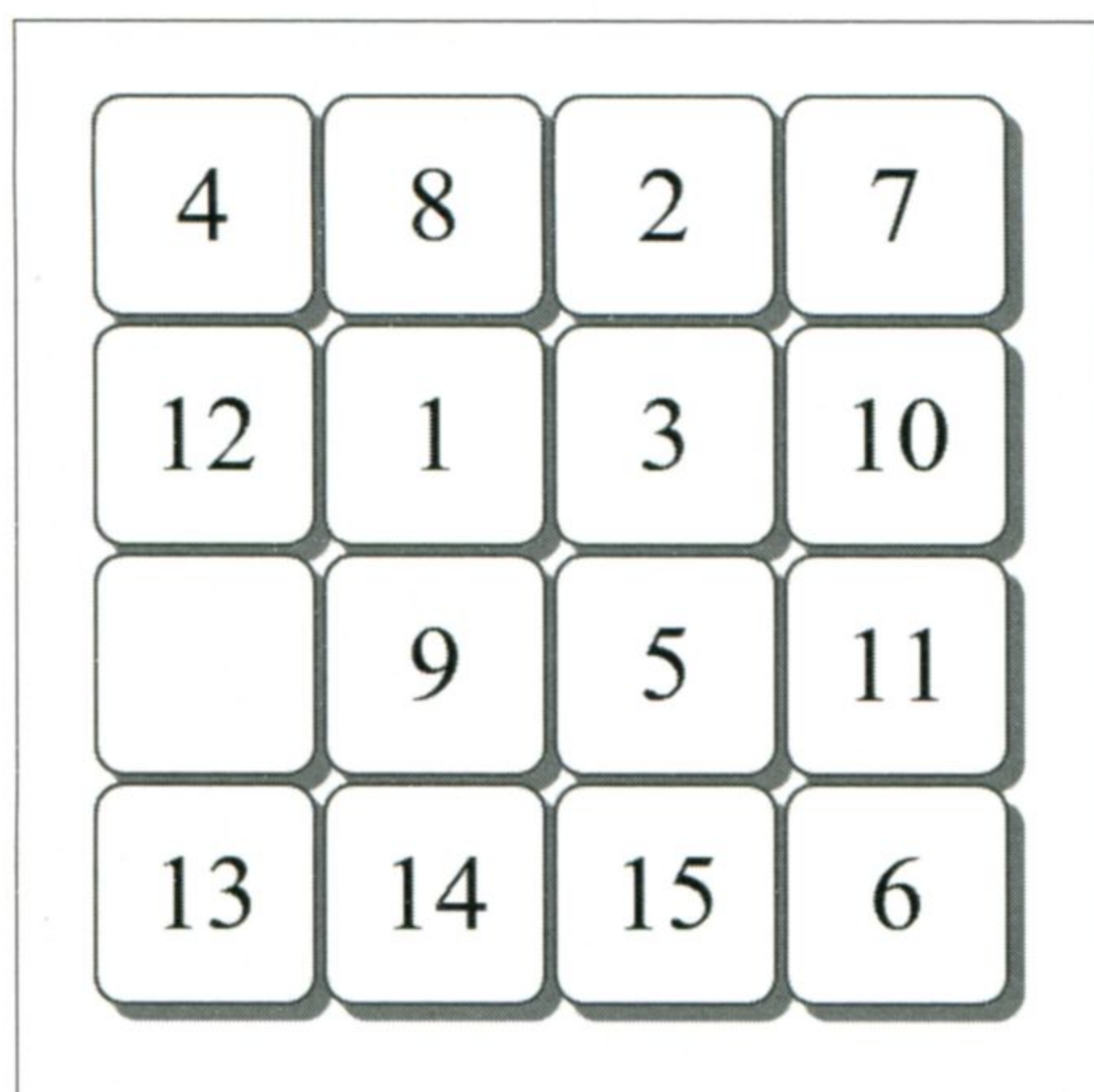
空欄の上下左右にあるタイルをクリックするとそのタイルが移動します。1～15まで順番になるように並べるゲームです。

4	8	2	7
12	1	3	10
	9	5	11
13	14	15	6

2	3		15
4	9	12	5
8	7	6	10
13	1	14	11

このゲームで学ぶこと

- DOM要素をJavaScriptから作成する
- for文の二重ループになれる

Let's write codes !

```
<!DOCTYPE html>
<html>
<head>
    <title>15 puzzle</title>
    <meta charset="UTF-8">
    <style>
        .tile {
            width: 70px;
            height: 70px;
            border: 1px solid blue;
            border-radius: 10px;
            text-align: center;
            font-size: 36px;
            background-color: white;
            box-shadow: rgb(128, 128, 128) 5px 5px;
        }
    </style>
    <script>
        "use strict";          ←1

        // 広域変数
        var tiles = [];

        // 初期化関数
        function init() {          ←2
            var table = document.getElementById("table");     ←3

            for (var i = 0 ; i < 4 ; i++) {          ←4
                var tr = document.createElement("tr");     ←5
                for (var j = 0 ; j < 4 ; j++) {          ←6
                    var td = document.createElement("td");     ←7
                    var index = i * 4 + j;
                    td.className = "tile";
                    td.index = index;
                    td.value = index;
                    td.textContent = index == 0 ? "" : index;
                    td.onclick = click;
                    tr.appendChild(td);          ←8
                    tiles.push(td);
                }
                table.appendChild(tr);
            }
```

```
        for (var i = 0 ; i < 1000 ; i++) {
            click({ srcElement: {index: Math.floor(Math.random() * 16)}})
        }
    }

    function click(e) {          ←9
        var i = e.srcElement.index;          ←10

        if (i - 4 >= 0 && tiles[i - 4].value == 0) {
            swap(i, i - 4);
        } else if (i + 4 < 16 && tiles[i + 4].value == 0) {
            swap(i, i + 4);
        } else if (i % 4 != 0 && tiles[i - 1].value == 0) {     ←11
            swap(i, i - 1);
        } else if (i % 4 != 3 && tiles[i + 1].value == 0) {
            swap(i, i + 1);
        }
    }

    function swap(i, j) {
        var tmp = tiles[i].value;
        tiles[i].textContent = tiles[j].textContent;
        tiles[i].value = tiles[j].value;
        tiles[j].textContent = tmp;
        tiles[j].value = tmp;
    }
  </script>
</head>
<body onload="init()">
  <table id="table"></table>
</body>
</html>
```

5-1-1 ソースコード解説

1の「"use strict"」を記述しておくと、より厳密にエラーチェックが行われます。潜在的なバグを軽減させるためにも記述しておくことをお勧めします。

NOTE JavaScriptはもともと、変数を宣言しなくても利用できたり、同じ名前の関数を重複して宣言できたり、とゆるい言語仕様を採用していました。しかし、このようなゆるい仕様は、習得の敷居を低くする一方で、バグの温床にもなってしまいました。そこで、より厳密にエラーチェックをするために「"use strict"」という運用が定められました。この行を記載しておくと、より厳密なエラーチェックが行われるようになります。バグを減らすためにも、この行を挿入する習慣をつけておくとよいでしょう。

2のinit()では初期化を行っています。まず、3の「var table = document.getElementById("table")」でtable要素への参照を取得しています。

行はtr要素で、その中のタイルはtd要素で実装しています。外側のfor文4を4回繰り返すことで4行作成しています。各行は5の「var tr = document.createElement("tr")」で作成しています。

1行作成するたびに、内側のfor文6でタイルを4つ作成しています。各タイルは7の「var td = document.createElement("td")」で作成し、8のtr.appendChild(td)で行に挿入しています。要は4×4の盤面をJavaScriptから作成しているだけです。

value=0のタイルは何も描画しない空タイルです。今回のポイントはindexとvalueプロパティの使い方です。indexはタイルの並び順で、valueはタイルに描画されている数値です。このふたつを混乱しないように注意してください。

indexはタイルの並び順、valueはタイルに描画されている数値

index=0 3 value=3	index=1 10 value=10	index=2 7 value=7	index=3 1 value=1
index=4 6 value=6	index=5 11 value=11	index=6 2 value=2	index=7 value=0
index=8 8 value=8	index=9 9 value=9	index=10 14 value=14	index=11 13 value=13
index=12 12 value=12	index=13 4 value=4	index=14 15 value=15	index=15 5 value=5

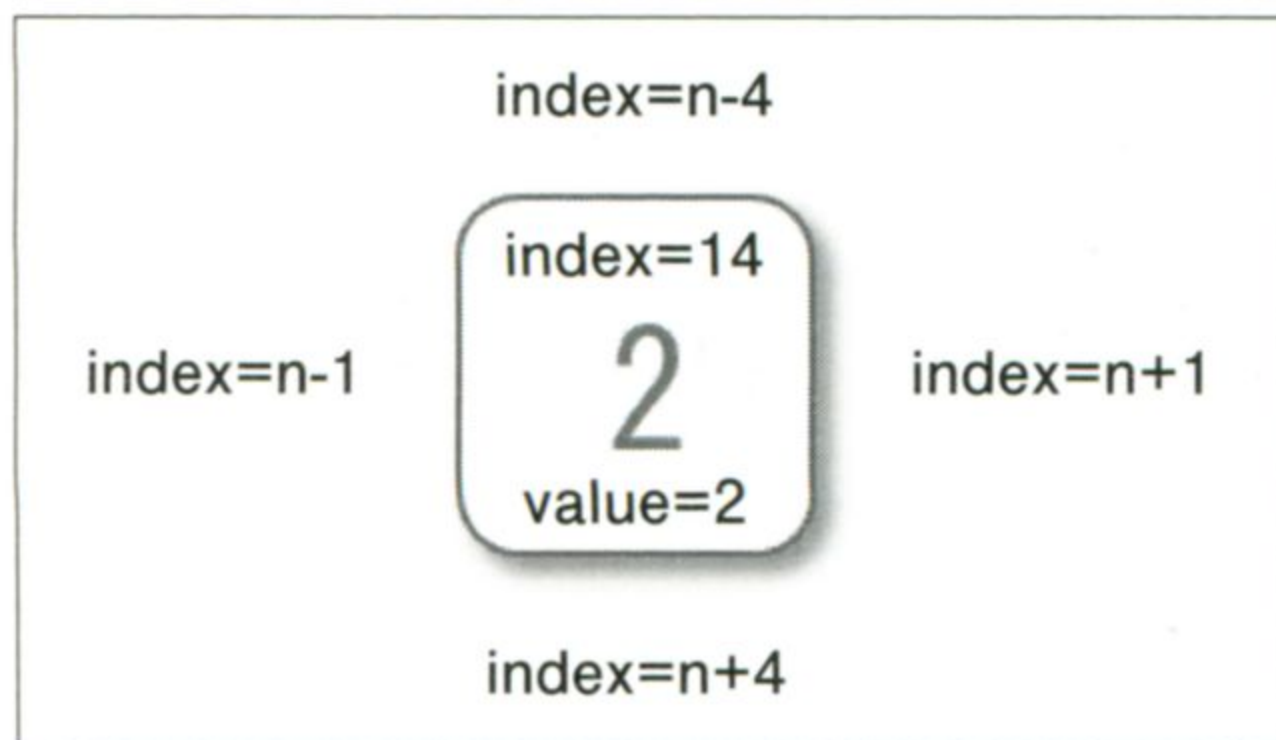
タイルがクリックされると、9のclick(e)が呼び出されます。そのタイルの上下左右のどこかに空タイル(valueプロパティが0のタイル)があった場合、それらタイルのvalueを入れ替えます。そのためには、クリックされたタイルの上下左右に空タイルがあるか調べる必要があります。

クリックしたタイルのindexは10のe.srcElement.indexで取得できます。よって、上のタイルはindex-4、下のタイルはindex+4、左のタイルはindex-1、右のタイルはindex+1で求められます。

クリックしたタイルとその上下左右のタイルのindex

index=n-4

index=n-1 | index=14 2 value=2 | index=n+1

index=n+4

ただし、値を比較する際には若干の注意が必要です。たとえば最上段がクリックされた場合、その上に行はないので比較できません。逆に最下段がクリックされた場合、その下に行はありません。

これらの処理はすべてclick(e)の中で行われています。たとえば、上のタイルとの比較は以下のように行っています（10の先頭のif文）。

```
if (i - 4 >= 0 && tiles[i - 4].value == 0) {
    swap(i, i - 4);
}
```

index-4が0以上であるかを最初にチェックし、それがtrueの場合、すなわち上にタイルが存在する場合に限り、「tiles[i - 4].value」の値が0か比較しています。

4つのif文は上下左右の比較を行っているので見比べてみるとその意図がわかると思います。条件式が成立した場合は「swap(i, j)」を呼び出してタイルの番号を入れ替えています。textContent（画面表示用）とvalue（内部管理用）のプロパティを入れ替えていることに注意してください。

ちなみに、以下の行はタイルがランダムに1000回クリックされた状況を再現しています。

```
for (var i = 0 ; i < 1000 ; i++) {
    click({ srcElement: {index: Math.floor(Math.random() * 16)}})
}
```

クリックされたときにclick(e)が呼び出されますが、click(e)の中ではe.srcElement.indexでインデックスを取得しています。その関数に合わせて引数を作成しています。click(e)を流用せずに別途処理を記述してもよかったのですが、全体の行数を削減するためにこのような実装としました。

演習　行と列を増やしてみよう

5行×5行のパズルに変更してみてください。

5-2 FlipCards

神経衰弱ゲームです。遊び方の説明は不要でしょう。すべてのペアを揃えるまでの経過時間が表示されます。

このゲームで学ぶこと

- Arrayオブジェクトのprototypeを使ってみる
- タイマーの使い方になれる

Let's write codes !

```
<!DOCTYPE html>
<html>
<head>
    <title>Flip Cards</title>
    <META charset="UTF-8">
    <style>
        td.card {           ←1
            width: 100px;
            height: 140px;
            border: 1px solid blue;
            border-radius: 10px;
            text-align: center;
            font-size: 36px;
            background-color: white;
            box-shadow: rgb(128, 128, 128) 5px 5px;
        }
        td.back {           ←2
            background-image: url("card.png");        ←3
            background-size: 100px 140px;
        }
    </style>
    <script>
        "use strict";
        // 配列シャッフル
        Array.prototype.shuffle = function () {        ←4
            var i = this.length;
            while (i) {
                var j = Math.floor(Math.random() * i);
                var t = this[--i];
                this[i] = this[j];
                this[j] = t;
            }
            return this;
        }

        // 広域変数
        var timer = NaN, score = 0, flipTimer, prevCard, startTime;

        // 初期化関数
        function init() {        ←5
            var table = document.getElementById("table");

            var cards = [];
                                                  ←6
```

```
    for (var i = 1 ; i <= 10 ; i++) {
        cards.push(i);
        cards.push(i);
    }
    cards.shuffle();

    for (var i = 0 ; i < 4 ; i++) {          ←7
        var tr = document.createElement("tr");
        for (var j = 0 ; j < 5 ; j++) {
            var td = document.createElement("td");
            td.className = "card back";
            td.number = cards[i * 5 + j];
            td.onclick = flip;
            tr.appendChild(td);
        }
        table.appendChild(tr);
    }

    startTime = new Date();
    timer = setInterval(tick, 1000);
}

// 経過時間計測用タイマー
function tick() {          ←8
    var now = new Date();
    var elapsed = Math.floor((now.getTime() - startTime.getTime()) / 1000);
    document.getElementById("time").textContent = elapsed;
}

// カード裏返し
function flip(e) {
    var src = e.srcElement;
    if (flipTimer || src.textContent != "") {          ←10
        return;
    }

    var num = src.number;
    src.className = "card";          ←11
    src.textContent = num;          ←9

    // 1枚目
    if (prevCard == null) {
        prevCard = src;          ←12
        return;
    }
```

```
            // 2枚目
            if (prevCard.number == num) {
                if (++score == 10) {
                    clearInterval(timer);
                }
                prevCard = null;
                clearTimeout(flipTimer);                    ←13
            } else {
                flipTimer = setTimeout(function () {
                    src.className = "card back";
                    src.textContent = "";
                    prevCard.className = "card back";
                    prevCard.textContent = "";              ←14
                    prevCard = null;
                    flipTimer = NaN;
                }, 1000);
            }
        }                                                              ←9 ここまで
    </script>
</head>
<body onload="init()">
    <table id="table"></table>
    <h2>
        経過時間：<span id="time">0</span>秒
    </h2>
</body>
</html>
```

5-2-1 ｜ソースコード解説

CSSから見ていきましょう。<style>要素では**1** td.card（カードの大きさ、フォントサイズなど）と**2** td.back（裏面）というふたつのセレクタを定義しています。JavaScriptでは、これらのスタイルを動的に適用するために以下のような処理を行っています。

```
element.className = "card back";     // カードが裏返された描画
element.className = "card"; // カードが表に返された描画
```

ここで"element"はtd要素を参照する変数です。プログラムの中ではtdという変数やsrcという変数が使用されています。

class属性をJavaScriptから設定するときは、classNameプロパティに値を代入します。代入する値を空白で区切ることにより、複数のセレクタを一度に設定していることに注目してください。

ちなみに、3の「background-image: url("card.png")」は背景画像を設定するCSSスタイルです。

4のArray.prototype.shuffleは配列の要素をランダムに入れ替えるためのメソッドです。プロトタイプのところで詳しく説明しています。忘れた人は読み直してください。

使用している広域変数は以下のとおりです。

使用している広域変数

変数	説明
timer	1秒ごとにtick()を呼び出すためのタイマー
flipTimer	2枚目にめくったカードをしばらく表示状態にしておくためのタイマー
score	何ペアー致したか
prevCard	1枚目にめくったカード
startTime	最初にゲームを開始した時刻

このゲームでは2つのタイマーを使っていることに注意してください。timerは経過時間を計測するためのものでsetInterval、clearIntervalで制御されます。一方、flipTimerは1秒後にカードを裏返すためのもので、setTimeout、clearTimeoutで制御されます。混乱しないように注意してください。

例によって、処理はinit()関数から始まります5。

6では20枚（ペア×10組）のカードをランダムに並べ替えています。

```
var cards = [];
for (var i = 1 ; i <= 10 ; i++) {
    cards.push(i);
    cards.push(i);
}
cards.shuffle();
```

そのあとの7の二重のfor文でカードを並べています。15パズルと同じ処理なので説明は省略します。

8のtick()関数では、経過時間を表示しています。

```
function tick() {
    var now = new Date();    ←Ⓐ
    var elapsed = Math.floor((now.getTime() - startTime.getTime()) / 1000); ←Ⓑ
    document.getElementById("time").textContent = elapsed;
}
```

Ⓐのnew Date()で現在時刻が取得できます。

Ⓑでは、Dateオブジェクトのget Time()メソッドで、1970年1月1日00:00:00UTCからの経過ミリ秒を取得しています。ゲーム開始時の値との差分をとり、1000で割ることにより秒単位の経過時間を求め、画面に表示しています。

リスト全体に戻りましょう。9のflip(e)以下が肝となる部分です。カードがクリックされたときに呼び出されるイベントハンドラですが、状況に応じて処理内容が異なります。順を追って見ていきましょう。

まず、10の処理です。

```
function flip(e) {
    var src = e.srcElement;
    if (flipTimer || src.textContent != "") {
        return;
    }
```

flipTimerが値を保持している間（2枚が表になってしばらく数字が表示されている間）、もしくはすでに表になったカードがクリックされた場合（「src.textContent != ""」）は、何も行わずにreturnで戻ります。

次に11の、

```
var num = src.number;
src.className = "card";
src.textContent = num;
```

でクリックされたカードを表にします。

次の12の、

```
// 1枚目
if (prevCard == null) {
    prevCard = src;
    return;
}
```

は「prevCard == null」が成り立つ場合、すなわち、今クリックされたカードが1枚目だったときは、現在のカードをprevCardに代入し、returnで関数を抜けます。2枚目のクリックを待つためです。

それ以外のときは2枚目がクリックされたときとなります。1枚目と同じカードのときは13の処理が行われます。10枚目になったときは全部のカードが裏返しになったので経過時間を計測するためのtimerを止めます。2枚のカードが同じだったので、prevCardにnull代入するとともに（=1枚目をクリア）、元に裏返すためのタイマーflipTimerを停止させます。

```
// 2枚目
if (prevCard.number == num) {
    if (++score == 10) {
        clearInterval(timer);
    }
    prevCard = null;
    clearTimeout(flipTimer);
}
```

1枚目と2枚目が異なるときは14の処理が行われます。

```
flipTimer = setTimeout(function () {
    src.className = "card back";
    src.textContent = "";
    prevCard.className = "card back";
    prevCard.textContent = "";
    prevCard = null;
    flipTimer = NaN;
}, 1000);
```

1秒後に無名関数が実行されます。つまり、1秒間ふたつの数字が表示された状態となります。無名関数の中では1枚目のカードprevCardと2枚目のカードsrcに対して、classNameでスタイルを適用し、textContentに空文字を代入することで元の裏返しの状態に戻しています。また、prevCardとflipTimerの値をクリアして、最初の状態に戻しています。

神経衰弱ゲームの解説は以上です。シンプルなゲームではありますが、1枚目をめくっているのか、2枚目をめくっているのかなど場合分けが少々面倒に感じられたかもしれません。

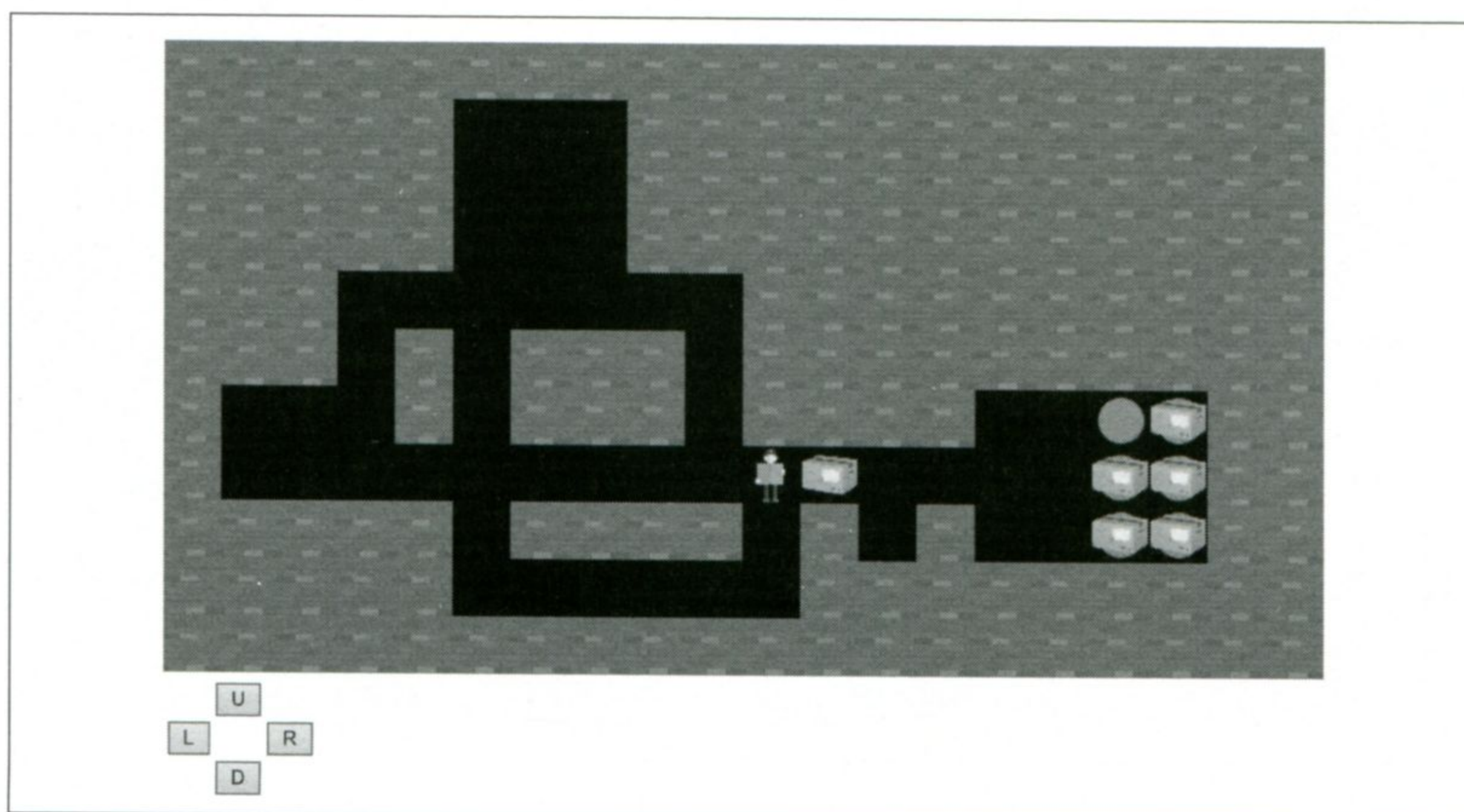
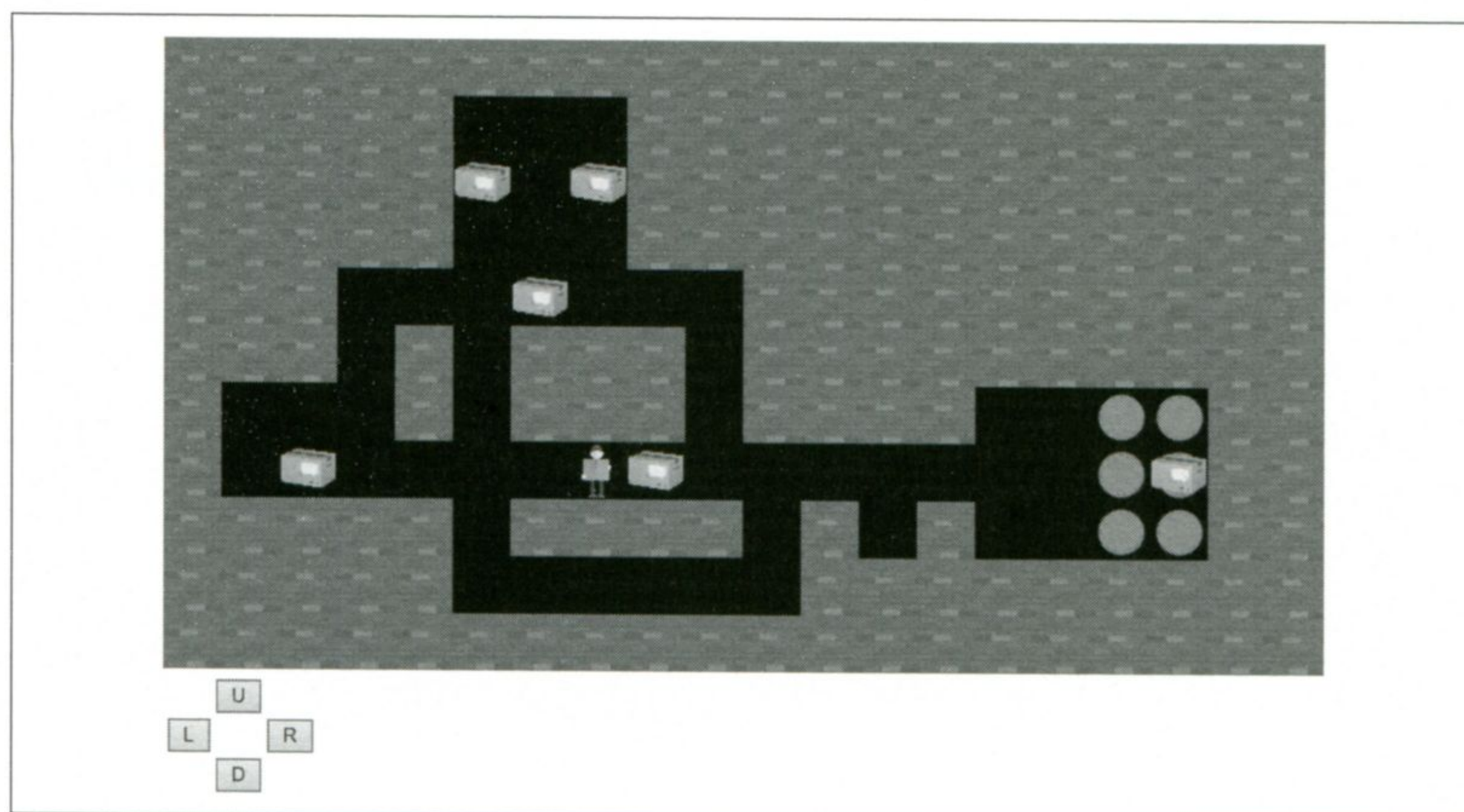
演習　枚数を変えてみよう

1 〜 13までのカードが4種類、計52枚の神経衰弱ゲームを作成してください。

5-3 CarryIt

昔からある定番ゲームです。人が荷物を所定の位置に移動させるだけのシンプルなゲームです。ただし、人は荷物を引っ張ることはできず、押すことしかできません。また、ふたつを同時に押すことはできません。すべての荷物を所定の場所に移動してください。実際にやってみると思いのほか難しいかもしれません。

U
L R
D

U
L R
D

矢印キーで人を動かして荷物を押していき、荷物を●印に収めればOK

このゲームで学ぶこと

- 仮想マップの使い方になれる
- JavaScriptから画像をcanvasに描画する
- ビット演算になれる

Let's write codes !

```
<!DOCTYPE html>
<html>
<head>
    <title>CarryIt</title>
    <meta charset="UTF-8">
    <script>
        "use strict";
        var data = [
            [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6],
            [6, 6, 6, 6, 6, 0, 0, 0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6],
            [6, 6, 6, 6, 6, 2, 0, 0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6],
            [6, 6, 6, 6, 6, 0, 0, 2, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6],
            [6, 6, 6, 0, 0, 2, 0, 0, 2, 0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6],
            [6, 6, 6, 0, 6, 0, 6, 6, 6, 0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6],   ←1
            [6, 0, 0, 0, 6, 0, 6, 6, 6, 0, 6, 6, 6, 6, 0, 0, 1, 1, 6, 6],
            [6, 0, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 6, 6],
            [6, 6, 6, 6, 6, 0, 6, 6, 6, 6, 0, 6, 0, 6, 0, 0, 1, 1, 6, 6],
            [6, 6, 6, 6, 6, 0, 0, 0, 0, 0, 0, 6, 6, 6, 6, 6, 6, 6, 6, 6],
            [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6],
        ];
        var gc, px = 12, py = 8;

        /**
         * 初期化関数
         */
        function init() {   ←2
            gc = document.getElementById("soko").getContext("2d");
            onkeydown = mykeydown;
            repaint();
        }

        function u(){ mykeydown({keyCode:38}); }
        function d(){ mykeydown({keyCode:40}); }   ←3
        function l(){ mykeydown({keyCode:37}); }
        function r(){ mykeydown({keyCode:39}); }

        function mykeydown(e) {   ←4
            var dx0 = px, dx1 = px, dy0 = py, dy1 = py;
            switch (e.keyCode) {
                case 37: dx0--; dx1-=2;
                    break;   ←5
                case 38: dy0--; dy1-=2;
                    break;
```

```
            case 39: dx0++; dx1+=2;
                break;
            case 40: dy0++; dy1+=2;
                break;
        }

        if ((data[dy0][dx0] & 0x2) == 0) {  // 荷物なし&壁なし→進む
            px = dx0;
            py = dy0;
        } else if ((data[dy0][dx0] & 0x6) == 2) { // 進行方向に荷物あり
            if ((data[dy1][dx1] & 0x2) == 0) {  // 荷物なし&壁なし→進む
                data[dy0][dx0] ^= 2;    // 隣の荷物をクリア                ←6
                data[dy1][dx1] |= 2;    // 更に先に荷物をセット
                px = dx0;
                py = dy0;
            }
        }

        repaint();
    }

    function repaint() {     ←7
        gc.fillStyle = "black";
        gc.fillRect(0, 0, 800, 440);

        for (var y = 0 ; y < data.length ; y++) {
            for (var x = 0 ; x < data[y].length ; x++) {
                if (data[y][x] & 0x1) {
                    gc.drawImage(imgGoal, x * 40, y * 40, 40, 40);
                }
                if (data[y][x] & 0x2) {
                    gc.drawImage(imgLuggage, x * 40, y * 40, 40, 40);
                }
                if (data[y][x] == 6) {
                    gc.drawImage(imgWall, x * 40, y * 40, 40, 40);
                }
            }
        }
        gc.drawImage(imgWorker, px * 40, py * 40, 40, 40);
    }
  </script>
</head>
<body onload="init()">
  <canvas id="soko" width="800" height="440"></canvas>
<table>
```

```
    <tr><td></td><td><button onclick="u()">U</button></td><td></td></tr>
    <tr><td><button onclick="l()">L</button></td><td></td>
        <td><button onclick="r()">R</button></td></tr>
    <tr><td></td><td><button onclick="d()">D</button></td><td></td></tr>
</table>
    <img id="imgWall" src="imgWall.png" style="display:none" />
    <img id="imgGoal" src="imgGoal.png" style="display:none" />
    <img id="imgWorker" src="imgWorker.png" style="display:none" />
    <img id="imgLuggage" src="imgLuggage.png" style="display:none" />
</body>
</html>
```

5-3-1 ソースコード解説

使用している広域変数は以下のとおりです。

使用している広域変数

変数	説明
gc	canvasに描画するためのグラフィックコンテキスト
px	主人公のx座標
py	主人公のy座標
data	地図データ、0:通路，1:目的地，2:荷物，6:壁

▶2 init()

初期化関数です。canvasの描画用コンテキストを変数gcに格納し、キー押下時のイベントハンドラにmykeydownを登録し、再描画をしています。

▶3 u(),d(),l(),r()

上下左右ボタンが押されたときのコールバックです。<table>要素を見ていただければその中に上下左右ボタンが配置され、上下左右ボタンのonclick属性が設定されていることがわかります。対応するキーが押下されたときの処理を行っています。

▶4 mykeydown(e)

キー押下時のイベントハンドラです。キー押下時のイベントハンドラでは、引数eのkeyCodeプロパティを見ることでどのキーが押されたかがわかります5。このゲームでは、移動方向の先に荷物があるか、さらにその先

が空いているか、とふたつ先まで調べる必要があります。それらの座標を管理するための変数がdx0、dy0、dx1、dy1です。ひとつ先の座標が(dx0、dy0)で、さらに先の座標が(dx1、dy1)です。その様子を以下の図に示します。

キー押下時のイベントハンドラ

ここから先のコードを理解するには2進数と論理演算の理解が必要です。実は、この程度のゲームであれば、論理演算を使わなくても十分実装可能です。しかし、論理演算を使用することでコードの行数を削減できそうだったのと、論理演算を学ぶよい機会だと思ったので使用することにしました。では、なぜ論理演算を使うと便利なのかその理由を考えてみましょう。

ゲームの状態はdataという二次元配列で管理しています**1**。ここで問題になるのは荷物が目的地に重なった場合です。

荷物と目的地の状態

0	0	1	1
0	2	1	1
0	0	1	1

0	0	1	1
0	0	2	1
0	0	1	1

0	0	1	1
0	2	?	1
0	0	1	1

上図左の状態から荷物を右に動かすと、移動先の座標の値は2になります（上図中央）。さらに回り込んで荷物を左に戻したとします（上図右）。すると荷物があった座標にはどんな値を設定すればよいでしょうか？ 目的地の1が荷物の2で上書きされてしまったので、1を設定するのか0を設定するのかわからなくなってしまいます。上書きする前の値を覚えておけば対処することも可能ですが処理が面倒です。

そこで、荷物があるか否かを単なる数字で表すのではなく、ビットの位置で表すことにしました。通路、目的地、荷物、壁に0、1、2、6といった数値を割り当てたのは実は意味があったのです。その理由はもう少し読み進めると理解できるので少々お待ちください。

では、論理演算について簡単に説明します。&はビットごとのAND演算、|はビットごとのOR演算を行う演算子です。ふたつの値（2進数1ビット）をAND、ORを計算した結果は以下のようになります。つまり、AND演算ではどちらも1のときのみ結果が1に、OR演算は少なくともどちらか1のときに結果が1になります。

AND演算

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

OR演算

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

10進数でAND演算、OR演算を計算する場合、いったん2進数に直して、それぞれのビットについて上記の表を適用します。たとえば、44と26のANDとORは次のようになります。

AND演算、OR演算の例

```
44 & 26 = 8            44 | 26 = 62

44 ➡101100             44 ➡101100
26 ➡011010             26 ➡011010
----------------       ----------------
    001000 ➡ 8             111110 ➡ 62
```

NOTE　ある状態を変数で管理する場合、取りうる値がどれかひとつの場合は単に数値を割り当てれば大丈夫です。たとえば、グー=0、チョキ=1、パー=2といった具合です。一方、取りうる状態が複数あるものを1つの変数で表現する場合、たとえば、朝・昼・晩それぞれの状態で、食べた・食べないといった値を表現するときは、ビット演算を使ったほうが便利です。1ビット目=朝、2ビット目=昼、3ビット目=夜としてしまえば、8とおりの状態を1つの変数で容易に表現できるようになります。

ここまでくればもう一息です。通路0、目的地1、荷物2、壁6と変な値を割り当てていましたが、実は荷物の有無と移動できるか否かを2ビット目が0か1かで判断していたのです。

まず、それぞれの値と2とのANDを計算してみましょう。0x2は2進数で010なので、この数値とANDを計算するということは、2ビット目を取り出すことと同じになります。その結果、通路と目的地は0、荷物と壁は1となりま

す。通路と目的地は自由に移動できます。つまり、データと0x2のAND計算結果が0であれば自由に動けるのです。

	10進数	2進数			& 0x2 (010とのAND)
		3ビット目	2ビット目	1ビット目	
通路	0	0	0	0	0
目的地	1	0	0	1	0
荷物	2	0	1	0	1
壁	6	1	1	0	1

次に、荷物が通路や目的地の上を動くことを考えます。荷物は0x2でした。通路や目的地とのORを取ることで、荷物のある通路、荷物のある目的地を表現できるようになります。

	2進数		10進数
	2ビット目	1ビット目	
通路	0	0	0
目的地	0	1	1
荷物 | 通路	1	0	2
荷物 | 目的地	1	1	3

準備はすべて整いました。**6**のコードを見てみましょう。

```
if ((data[dy0][dx0] & 0x2) == 0) {  // 通路か目的地→進む        ←Ⓐ
     px = dx0;
     py = dy0;
 } else if ((data[dy0][dx0] & 0x6) == 2) { // 進行方向に荷物あり   ←Ⓑ
     if ((data[dy1][dx1] & 0x2) == 0) {  // さらに先は通路か目的地  ←Ⓒ
         data[dy0][dx0] ^= 2;     // 隣の荷物をクリア
         data[dy1][dx1] |= 2;     // さらに先に荷物をセット     ←Ⓓ
         px = dx0;
         py = dy0;
     }
}
```

まず、Ⓐの「(data[dy0][dx0] & 0x2) == 0」という条件式により、ひとつ先が通路か目的地か判断しています。条件がtrueの場合は前に進めるので、現在地(px, py)をひとつ先の座標(dx0, dy0)に設定します。

次のif文の条件式Ⓑ「(data[dy0][dx0] & 0x6) == 2」は少し説明が必要です。なぜ0x2でなく0x6を使っているのでしょうか? 壁と荷物に対して0x2のAND演算を行うと、

- 壁：6 & 0x2 = 2
- 荷物：2 & 0x2 = 2

となります。壁と荷物は区別して処理することが必要なのに、壁と荷物の区別ができなくなってしまいます。よって、0x6（2進数では110）とのANDを取ることにします。

- 壁：6 & 0x6 = 6
- 荷物：2 & 0x6 = 2

これで、壁か荷物かを正しく区別できました。

次のif文Ⓒは、さらに先が移動可能かを判定するもので、最初のif文とほぼ同じです。次の場所に荷物があり、さらにひとつ先が移動可能ということは、荷物を移動できるということなので、Ⓓでその処理を行います。

```
data[dy0][dx0] ^= 2;     // 隣の荷物をクリア
data[dy1][dx1] |= 2;     // さらに先に荷物をセット
```

^演算子はビット排他論理和といって、0を1に、1を0に反転するものです。「^= 2」とすることで、2ビット目を反転した結果を自分自身に代入します。

｜演算子はOR演算子です。「｜= 2」とすることで、2ビット目のビットを1にした結果を自分自身に代入します。これらの処理を行うことで、隣にあった荷物をさらにひとつ先に移動しています。

このように論理演算を使用することで、データの様子は以下のようになります。

0	0	1	1
0	2	1	1
0	0	1	1

0	0	1	1
0	0	3	1
0	0	1	1

0	0	1	1
0	2	1	1
0	0	1	1

▶7 repaint()

まず以下のコードで背景をすべてクリアしています。

```
gc.fillStyle = "black";
gc.fillRect(0, 0, 800, 440);
```

あとは、二重のfor文でそれぞれのマスに応じた画像を描画しています。目的地の上に荷物を置いたりすることもあるので、else ifやswitchを使っていないことに注意してください。

最後にHTMLの部分です。「<canvas id="soko" width="800" height="440"></canvas>」はwidth/height属性を明示的に指定していることに注意してください。この指定を忘れると意図した大きさで描画されないことがあります。

「」といった画像はcanvasに描画するためのものです。HTMLで表示する必要はないので、「style="display:none"」とstyle属性を指定しています。

以上でこのゲームの説明は終了です。コードの行数は少ないものの、論理演算やcanvasへの描画などさまざまな要素が含まれています。ぜひしっかりと理解するようにしてください。

演習　自作パズルに挑戦してみよう

自分なりのパズル（マップ）を考えてみましょう。友人と問題の出し合いをしてもおもしろいでしょう。

5-4 Reversible Piece

これもルール説明不要の定番ゲームです。あまり強くはありませんがコンピュータがちゃんとあなたの相手をしてくれます。

黒（あなた）：4枚　白（PC）：4枚

黒（あなた）：29枚　白（PC）：13枚

このゲームで学ぶこと

- コンピュータの思考回路を実装する

Let's write codes !

```
<!DOCTYPE html>
<html>
<head>
    <title>ReversiblePiece</title>
    <meta charset="UTF-8">
    <style>
        #board {
            background-color:#555555;
        }
        td.cell {
            background-color: #FFB27F;
            width: 60px;
            height: 60px;
            margin: 2px;
            font-size: 50px;
            text-align: center;
        }
        td.black {
            color: black;
        }
        td.white {                              ←31
            color: white;
        }
    </style>
    <script>
        "use strict";

        var WeightData = [
            [ 30,-12,  0, -1, -1,  0, -12,  30],
            [-12,-15, -3, -3, -3, -3, -15, -12],
            [  0, -3,  0, -1, -1,  0,  -3,   0],
            [ -1, -3, -1, -1, -1, -1,  -3,  -1],
            [ -1, -3, -1, -1, -1, -1,  -3,  -1],      ←32
            [  0, -3,  0, -1, -1,  0,  -3,   0],
            [-12,-15, -3, -3, -3, -3, -15, -12],
            [ 30,-12,  0, -1, -1,  0, -12,  30],
        ];
        var BLACK = 1, WHITE = 2;
        var data = [];
        var myTurn = false;

        /**
         * 初期化関数
```

```
*/
function init() {                    ←1
    var b = document.getElementById("board");
    for (var i = 0 ; i < 8 ; i++) {
        var tr = document.createElement("tr");
        data[i] = [0, 0, 0, 0, 0, 0, 0, 0];
        for (var j = 0 ; j < 8 ; j++) {
            var td = document.createElement("td");
            td.className = "cell";                          ←2
            td.id = "cell" + i + j;   ←30
            td.onclick = clicked;     ←31
            tr.appendChild(td);
        }
        b.appendChild(tr);
    }
    put(3, 3, BLACK);
    put(4, 4, BLACK);
    put(3, 4, WHITE);                                        ←3
    put(4, 3, WHITE);
    update();
}

function update() {                  ←4
    var numWhite = 0, numBlack = 0;
    for (var x = 0 ; x < 8 ; x++) {
        for (var y = 0 ; y < 8 ; y++) {
            if (data[x][y] == WHITE) {
                numWhite++;
            }
            if (data[x][y] == BLACK) {                            ←5
                numBlack++;
            }
        }
    }
    document.getElementById("numBlack").textContent = numBlack;
    document.getElementById("numWhite").textContent = numWhite;

    var blackFlip = canFlip(BLACK);                          ←6
    var whiteFlip = canFlip(WHITE);

    if (numWhite + numBlack == 64 || (!blackFlip && !whiteFlip)) {
        showMessage("ゲームオーバー")                                   ←7
    }
```

```
    else if (!blackFlip) {
        showMessage("黒スキップ");
        myTurn = false;
    }
    else if (!whiteFlip) {
        showMessage("白スキップ");
        myTurn = true;
    }                                          ←8
    else {
        myTurn = !myTurn;
    }
    if (!myTurn) {
        setTimeout(think, 1000);
    }
}

function showMessage(str) {       ←9
    document.getElementById("message").textContent = str;
    setTimeout(function () {
        document.getElementById("message").textContent = "";
    }, 2000);
}

/**
 * 盤上のセルクリック時のコールバック関数
 */
function clicked(e) {       ←10
    if (!myTurn) {   // PC考え中
        return;                    ←11
    }
    var id = e.target.id;       ←12
    var i = parseInt(id.charAt(4));
    var j = parseInt(id.charAt(5));

    var flipped = getFlipCells(i, j, BLACK)
    if (flipped.length > 0) {
        for (var k = 0 ; k < flipped.length ; k++) {     ←13
            put(flipped[k][0], flipped[k][1], BLACK);
        }
        put(i, j, BLACK);
        update();
    }
}

/**
```

```
 * (i,j)にcolor色の駒を置く
 */
function put(i, j, color) {
    var c = document.getElementById("cell" + i + j);
    c.textContent = "●";
    c.className = "cell " + (color == BLACK ? "black" : "white");    ←14
    data[i][j] = color;
}

/**
 * コンピュータ思考関数
 */
function think() {          ←15
    var highScore = -1000;
    var px = -1, py = -1;          ←16
    for (var x = 0 ; x < 8 ; x++) {
        for (var y = 0 ; y < 8 ; y++) {
            var tmpData = copyData();
            var flipped = getFlipCells(x, y, WHITE);
            if (flipped.length > 0) {
                for (var i = 0 ; i < flipped.length ; i++) {    ←17
                    var p = flipped[i][0];
                    var q = flipped[i][1];
                    tmpData[p][q] = WHITE;
                    tmpData[x][y] = WHITE;
                }
                var score = calcWeightData(tmpData);
                if (score > highScore) {
                    highScore = score;          ←18
                    px = x, py = y;
                }
            }
        }
    }

    if (px >= 0 && py >= 0) {
        var flipped = getFlipCells(px, py, WHITE)
        if (flipped.length > 0) {
            for (var k = 0 ; k < flipped.length ; k++) {
                put(flipped[k][0], flipped[k][1], WHITE);    ←19
            }
        }
        put(px, py, WHITE);
    }
```

```
    update();
}

/**
 * 重みづけ計算
 */
function calcWeightData(tmpData) {        ←20
    var score = 0;
    for (var x = 0 ; x < 8 ; x++) {
        for (var y = 0 ; y < 8 ; y++) {
            if (tmpData[x][y] == WHITE) {
                score += WeightData[x][y];
            }
        }
    }
    return score;
}

/**
 * 駒テーブルデータをコピー
 */
function copyData() {        ←21
    var tmpData = [];
    for (var x = 0 ; x < 8 ; x++) {
        tmpData[x] = [];
        for (var y = 0 ; y < 8 ; y++) {
            tmpData[x][y] = data[x][y];
        }
    }
    return tmpData;
}

/**
 * 挟める駒があるか？
 */
function canFlip(color) {        ←22
    for (var x = 0 ; x < 8 ; x++) {
        for (var y = 0 ; y < 8 ; y++) {
            var flipped = getFlipCells(x, y, color);
            if (flipped.length > 0) {
                return true;
            }
        }
    }
    return false;
```

```
}

/**
 * (i,j)に駒をおいたときに駒を挟めるか？
 */
function getFlipCells(i, j, color) {          ←23
    if (data[i][j] == BLACK || data[i][j] == WHITE) {   // すでに駒がある
        return [];                                                    ←24
    }

    // 相手を挟めるか、左上、上、右上、左、右、左下、下、右下と順番に調査
    var dirs = [[-1,-1],[0,-1],[1,-1],[-1,0],[1,0],[-1,1],[0,1],[1,1]];
    var result = [];
    for (var p = 0 ; p < dirs.length ; p++) {
        var flipped = getFlipCellsOneDir(i, j,                        ←25
            dirs[p][0], dirs[p][1], color);
        result = result.concat(flipped)
    }
    return result;
}

/**
 * (i,j)に駒をおいたときに、(dx,dy)方向で駒を挟めるか？
 */
function getFlipCellsOneDir(i, j, dx, dy, color) {          ←26
    var x = i + dx;
                              ←27
    var y = j + dy;
    var fliped = [];

    if (x < 0 || y < 0 || x > 7 || y > 7 ||
        data[x][y] == color || data[x][y] == 0) {
        // 盤外、同色、空ならfalse                       ←28
        return [];
    }
    fliped.push([x, y]);     ←29

    while (true) {
        x += dx;
        y += dy;
        if (x < 0 || y < 0 || x > 7 || y > 7 || data[x][y] == 0) {
            // 盤外、空ならfalse                                    ←30
            return [];
        }
        if (data[x][y] == color) { // 挟めた！
            return fliped;
```

```
                } else {
                    fliped.push([x, y]);
                }
            }
        }

    </script>
</head>
<body onload="init()">
    <h2>
        黒（あなた）：<span id="numBlack"></span>枚
        白（ＰＣ）：<span id="numWhite"></span>枚
    </h2>
    <table id="board"></table>
    <h2 id="message"></h2>
</body>
</html>
```

5-4-1 ｜ ソースコード解説

使用している広域変数は以下のとおりです。

使用している広域変数

変数	説明
WeightData	重みづけ（どこに優先して石をおくか）計算用の配列
data	盤面における石の配置状態を保持する配列
BLACK = 1	盤面dataに置かれた石（黒）を1で表現
WHITE = 2	盤面dataに置かれた石（白）を2で表現
myTurn	自分の番か否か

今回は関数の数が多いので順番に見ていきましょう。

▶1 init()

15ゲームや神経衰弱と同じように8×8の盤を作ります。2で縦軸をi、横軸をjとして二重のfor文を使ってtd要素とdataを初期化しています。

二次元配列dataの様子を次に示します。石が置かれていない場所は0、黒が置かれた場所は1、白が置かれた場所は2として管理しています。たとえばdata[3][3]は黒なので、「data[3][3]=1」となります。

8×8のマス目をつくる

	J=0	J=1	J=2	J=3	J=4	J=5	J=6	J=7
i=0	data [0][0]	data [0][1]	data [0][2]	data [0][3]	data [0][4]	data [0][5]	data [0][6]	data [0][7]
i=1	data [1][0]	data [1][1]	data [1][2]	data [1][3]	data [1][4]	data [1][5]	data [1][6]	data [1][7]
i=2								
i=3								
i=4								
i=5								
i=6								
i=7	data [7][0]	data [7][1]	data [7][2]	data [7][3]	data [7][4]	data [7][5]	data [7][6]	data [7][7]

3では、最初の4つの石を配置してupdate()を呼び出しています。

▶4 update()

このゲームは人間とコンピュータが交互に石を配置していくのが基本ですが、石が置けない場合は相手に順番を譲る必要があります。また、すべて石が置かれたり、双方が同時に石を置けなくなったりするとゲームオーバーになります。このようにさまざまな状況に対処する必要がありますが、それを処理しているのがupdate()関数です。

update()関数は初期化直後や石を配置したときに呼び出されます。5で、盤に置かれた白の数numWhiteと黒の数numBlackを数え、「document.getElementById("numBlack").textContent=numBlack」と続く行で、枚数を画面に表示しています。

次の6の部分が興味深いところです。canFlipは引数の石を盤に置くことができるか（=ほかの色を挟むことができるか）を返す関数です22。たとえば、黒を配置できる状態であればblackFlipがtrueとなります。

```
var blackFlip = canFlip(BLACK);
var whiteFlip = canFlip(WHITE);
```

7では、ゲームオーバー、スキップの判定を行います。白と黒の数の合計が64になる、もしくは、白も黒も置けない場合はゲームオーバーとなります。

```
if (numWhite + numBlack == 64 || (!blackFlip && !whiteFlip)) {
    showMessage("ゲームオーバー")
}
```

8は、石を置く順番を判定する処理です。

```
else if (!blackFlip) {         ←Ⓐ
    showMessage("黒スキップ");
    myTurn = false;
}
else if (!whiteFlip) {         ←Ⓑ
    showMessage("白スキップ");
    myTurn = true;
}
else {
    myTurn = !myTurn;          ←Ⓒ
}
if (!myTurn) {                 ←Ⓓ
    setTimeout(think, 1000);
}
```

もし、blackFlipがfalseであればⒶ、黒が置けないので黒がスキップとなり相手の番（「myTurn = false」）となります。逆に、白が置けなければⒷ、自分の番（「myTurn = true」）となります。それ以外の場合はⒸ、順番を交代します。

「!」は論理否定を行う演算子で、trueはfalseに、falseはtrueに変換されます。たとえば、

```
myTurn=!myTurn;
```

のように記述すると、myTurnの値がtrue→false→true→falseと順番に変化します。覚えておくと便利な小技の一つです。

自分の番でないとき（!myTurn）はⒹ、コンピュータに1秒間考えるフリをさせます。

思考アルゴリズムがシンプルなため一瞬で処理が終わり、ゲームっぽくならないためです。

▶9 showMessage

引数で与えられた文字列を2秒間だけ表示します。

▶10 clicked(e)

この関数は1のinit関数でtd要素のイベントハンドラとして登録されたものです31。myTurnがfalseのとき(!myTurn)は、PCが思考中(のフリ)なので何もせずに戻ります11。

12の「var id = e.target.id」では、どのtdがクリックされたか調べるためのidを取得しています。これはinit関数の30で設定された内容です。

13の処理で縦・横の座標を求め、そこに黒を置いたときに反転する石を配列で求めています。

```
var i = parseInt(id.charAt(4));
var j = parseInt(id.charAt(5));
var flipped = getFlipCells(i, j, BLACK)
```

続くif文では、反転する石が0より多い場合、つまり反転する石があった場合、以下のコードでその石を反転させ、その場所に黒石を置いています。

```
if (flipped.length > 0) {
    for (var k = 0 ; k < flipped.length ; k++) {
        put(flipped[k][0], flipped[k][1], BLACK);
    }
    put(i, j, BLACK);
    update();
}
```

▶14 put(i, j, color)

座標(i, j)にcolor色の石を置きます。ポイントは以下の行で石の色を設定しているところです。

```
c.className = "cell " + (color == BLACK ? "black" : "white")
```

「(color == BLACK ? "black" : "white")」は三項演算子です(P.81「3-3-5 条件式 – 三項演算子」)。冒頭のCSS(style要素)の31には、以下のようなセレクタが宣言されています。JavaScriptからclassNameを動的に設定することで、セレクタを切り替えています。

```
td.black {
    color: black;
}
td.white {
    color: white;
}
```

つまり、以下のようにHTMLで記述したのと同じ効果を実現しています。

```
<td class="cell black">●</td>   ← 黒石
<td class="cell white">●</td>   ← 白石
```

▶15 think()

このゲームの肝となるコンピュータの思考ルーチンです。といっても処理内容はシンプルです。このゲームでは4隅を取ると圧倒的に有利になることはご存じでしょう。であれば、その4隅の周囲にはなるべく石を置かないほうがよいことは類推できると思います。このように盤面の場所ごとに優先度を設定し、その合計値が一番大きくなるような場所に石を配置するというのが今回実装したアルゴリズムです。場所ごとの優先度は広域変数WeightData 32 で設定しています。

場所ごとの優先度を変数で設定

30	-12	0	-1	-1	0	-12	30
-12	-15	-3	-3	-3	-3	-15	-12
0	-3	0	-1	-1	0	-3	0
-1	-3	-1	-1	-1	-1	-3	-1
-1	-3	-1	-1	-1	-1	-3	-1
0	-3	0	-1	-1	0	-3	0
-12	-15	-3	-3	-3	-3	-15	-12
30	-12	0	-1	-1	0	-12	30

コンピュータは白です。思考ルーチンの概要は以下のようになります。

① 64マス中で白を置ける場所を探し、そこに白石を置いた時の状態を再現する
② その状態において優先順位の合計値（白石のある場所の点の合計）を求める
③ 優先順位の合計値が最大になった場所に石を置く

順を追って見ていきましょう。まず、17では64マスのすべての場所をfor文の二重ループを使って調べます。

```
for (var x = 0 ; x < 8 ; x++) {
    for (var y = 0 ; y < 8 ; y++) {
        var tmpData = copyData();  ←Ⓐ
        var flipped = getFlipCells(x, y, WHITE);  ←Ⓑ
        if (flipped.length > 0) {
            for (var i = 0 ; i < flipped.length ; i++) {
                var p = flipped[i][0];
                var q = flipped[i][1];
                tmpData[p][q] = WHITE;
                tmpData[x][y] = WHITE;
            }
```

ⒶのcopyData()は現在のdataをコピーしたものを返します。これは「仮にこの場所に白をおいたときにどうなるか」という状態を再現するための仮データtmpDataを作るためです。

Ⓑの「var flipped = getFlipCells(x, y, WHITE)」で、(x, y) 座標に白を置いたときに反転する石の配列を求めます。石がある場合、つまり「(flipped.length > 0)」の場合は、仮データtmpDataに値を設定します。ここまでが前述の思考ルーチン①の部分です。

次に、18で合計点を計算します。これが思考ルーチン②です。

```
var score = calcWeightData(tmpData);
if (score > highScore) {
    highScore = score;
    px = x, py = y;
}
```

思考ルーチン③は、上のコードで「score > highScore」を比較している部分です。その値がハイスコアより大きい場合、すなわち今の打ち手のほうがより高得点のときは、現在の(x, y)座標をpxとpyに保存しておきます。think関数の冒頭16で、「px = −1, py = −1」と初期化しているので、両方が0以上ということは（）どこかに石を置けたことを意味します19。よって、pxとpyが0以上だったときには、その場所に白を置きます。

```
if (px >= 0 && py >= 0) {
    var flipped = getFlipCells(px, py, WHITE)
    if (flipped.length > 0) {
        for (var k = 0 ; k < flipped.length ; k++) {
            put(flipped[k][0], flipped[k][1], WHITE);
        }
    }
    put(px, py, WHITE);
}
```

以上がコンピュータの思考ルーチンです。

▶20 calcWeightData(tmpData)

引数で与えられた盤面データ（2次元配列）の優先順位の合計値を求めて返す関数です。白の置かれた場所の重みをscoreに足してその合計を返しています。

▶21 copyData()

優先順位の合計値を計算する場合には、仮に白を置いたときに、それによって石がひっくり返った状態を再現する必要があります。ひっくり返った石を元に戻しながら、都度すべての場所を調べる実装方法もありますが、処理が面倒になりそうだったので、現在の状態をコピーして使い捨てる手法をとることにしました。data配列をtmpData配列にコピーしているだけです。

▶22 canFlip(color)

盤面に引数の色の石を置けるか否かをtrueかfalseで返します。getFlipCells(x, y, color)関数を使って座標(x, y)に石を置いたときに反転する数を求め、それが0より大きい場合にtrueを返しています。

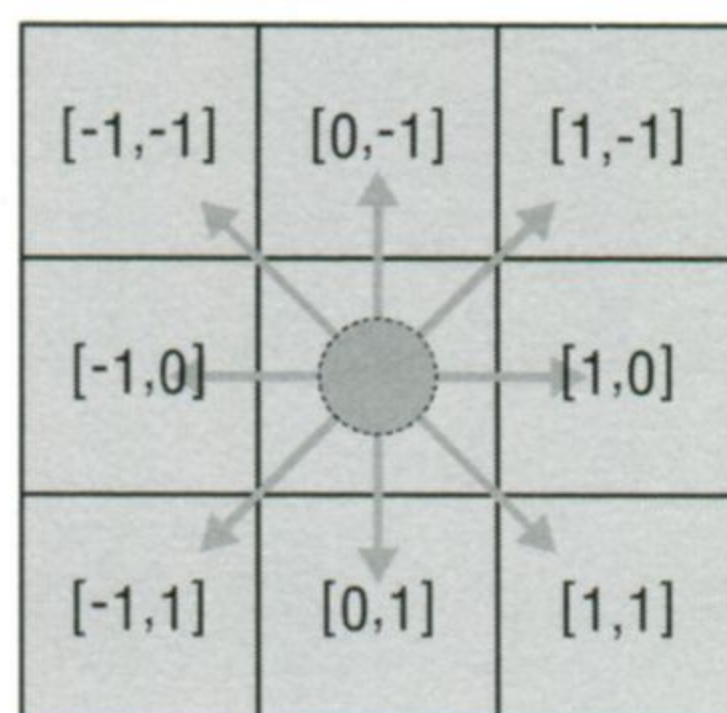
▶23 getFlipCells(i, j, color)

(i, j) 座標にcolorの石を置いたときに反転する石の配列を返します。

まず、すでに石が置かれている場合は挟めないので、24で空の配列[]を返します。

次の25は、石を置ける場合の処理です。石を置いたときに挟める方向は以下の図のように8方位となります。つまり、石を挟めるか否か判断するには8つの方位について調べる必要があるのです。

8つの方位

[-1,-1]	[0,-1]	[1,-1]
[-1,0]		[1,0]
[-1,1]	[0,1]	[1,1]

さらに、石を挟めるためには、「その隣に異なる色がひとつ以上並んでいて、かつ、その並びの先に自分と同じ色が存在する」必要があります。途中で石が置かれていなかったり、盤の外に出てしまったりするケースも考慮する必要があります。ひとつの関数ですべてを処理すると複雑になりそうだったので、ある方向で挟めるか判定する処理は別の関数getFlipCellsOneDir 26 に任せることにしました。

よって、この関数では、各方向を配列dirs格納し、for文を使って方向ごとに石を挟めるか否かを判定するだけにしています。

```
var dirs = [[-1, -1], [0, -1], [1, -1], [-1, 0], [1, 0], [-1, 1], [0, 1], [1, 1]];
var result = [];
for (var p = 0 ; p < dirs.length ; p++) {
    var flipped = getFlipCellsOneDir(i, j, dirs[p][0], dirs[p][1], color);
    result = result.concat(flipped)
}
return result;
```

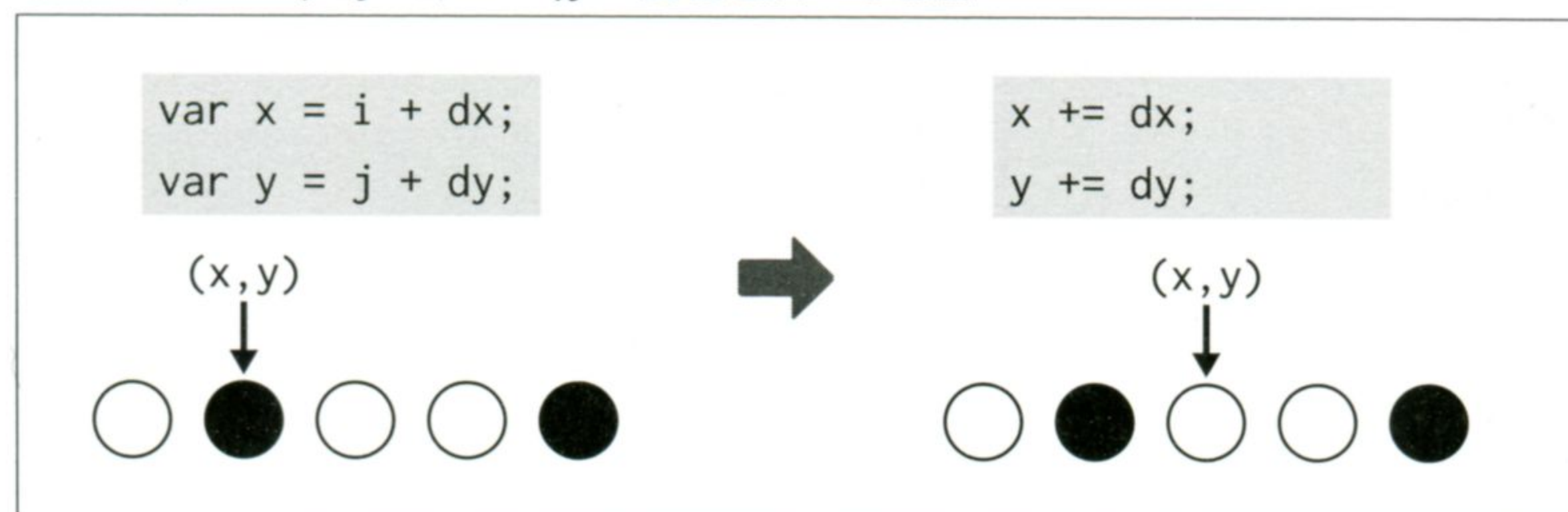
resultに反転した座標が格納されます。concatはArrayメソッドで、別の配列と結合する処理を行います。

▶ 26 getFlipCellsOneDir(i, j, dx, dy, color)

(i, j)を起点として、(dx, dy)方向に、color色の石で挟めるかを返します。順番に石を見て行く必要があるので、その座標を(x, y)としています 27。右方向に調べる場合、すなわち、dx=1, dy=0のときにx,yがどう変化するか、その様子を以下に示します。

右方向（dx=1, dy=0）にx,yの変化を調べる場合

```
var x = i + dx;
var y = j + dy;
```

```
x += dx;
y += dy;
```

まず、となりが同じ色、盤の外、石がないといった場合は単に空配列を返します28。

```
if (x < 0 || y < 0 || x > 7 || y > 7 ||
    data[x][y] == color || data[x][y] == 0) {
    // 盤外、同色、空からfalse
    return [];
}
```

そうでなければ、別の色の石がその方向に隣接していることになるので、29で、その石の座標を仮に配列flipedに格納しておきます。

```
fliped.push([x, y]);
```

あとは、30で、その方向を順番に見ていくことになります。

```
while (true) {
    x += dx;
    y += dy;
    if (x < 0 || y < 0 || x > 7 || y > 7 || data[x][y] == 0) {
        // 盤外、空ならfalse
        return [];
    }
    if (data[x][y] == color) { // 挟めた！
        return fliped;
    } else {
        fliped.push([x, y]);
    }
}
```

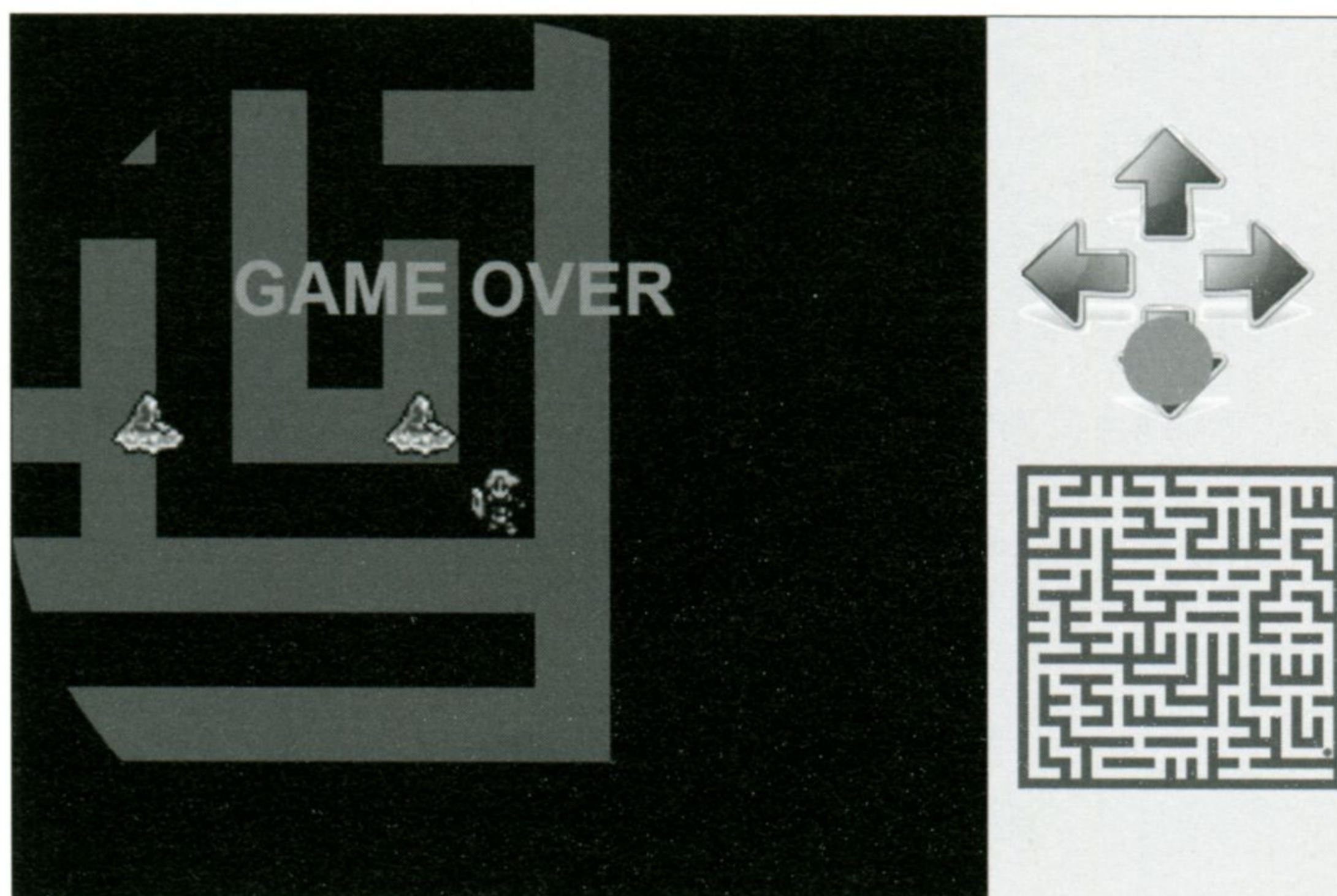
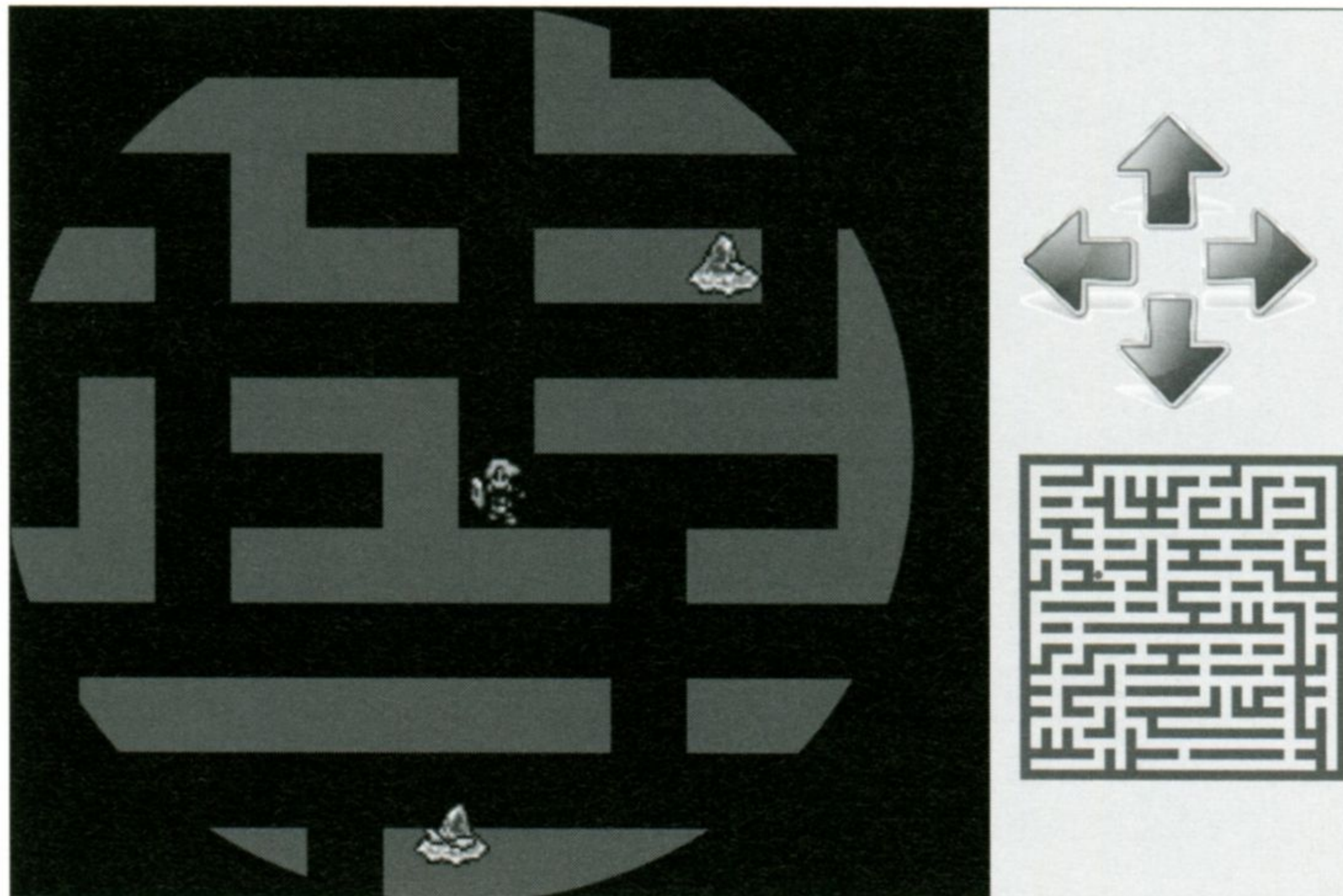
「x += dx」と「y += dy」でひとつ先に進めます。同様に盤外や空になったら挟めなかったということなので空配列を返します。その座標に同じ色があった場合は挟めたことになるので、配列flipedを返します。それ以外の場合は、別の色が連続していたことになるので、while文の実行を継続します。

このゲームの説明は以上です。「コンピュータと対戦する」という例として取り上げてみました。

演習　コンピュータを強くしてみよう

上の例では重みづけの配列WeightDataを固定として処理しています。実際には４隅をとったら、その周囲の優先度も高くなるはずです。つまり、WeightDataは状況に応じて途中で変化したほうがより強いアルゴリズムになるはずです。自分なりに工夫して、より強いコンピュータに育ててください。

5-5 Dungeon

「迷路のように入り組んだダンジョンを進んでモンスターと対決する」、よくあるゲームの１シーンです。そんな状況を再現するゲームをつくってみました。ちなみに迷路は毎回ランダムに作成されます。

GAME OVER

このゲームで学ぶこと

- 上下左右スクロールゲームに慣れる
- 迷路の自動生成を行う
- Canvasのリージョンクリップの手法を知る

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <style>
        #maze {
            width: 900px; height: 600px;
            touch-action: none;
        }
        #START {
            position: absolute;
            left: 200px;
            top: 200px;
        }
    </style>
    <script>
        "use strict";

        var W = 31, H = 31, GAMECLEAR = 1, GAMEOVER = 2;
        var ctx, keyCode = 0, maze = [], player, aliens = [];
        var timer = NaN, status = 0;

        var scroller = new Scroller();
        Player.prototype = scroller;
        Alien.prototype = scroller;

        /**
         * スクロール処理オブジェクト
         */
        function Scroller() {    ←1

            this.doScroll = function () {
                if (this.dx == 0 && this.dy == 0) {
                    return;
                }

                if (++this.scrollCount >= 5) {
                    this.x = this.x + this.dx;
                    this.y = this.y + this.dy;
                    this.dx = 0;
                    this.dy = 0;
                    this.scrollCount = 0;
```

```
            }
        }

        this.getScrollX = function () {
            return this.x * 50 + this.dx * this.scrollCount * 10;
        }

        this.getScrollY = function () {
            return this.y * 50 + this.dy * this.scrollCount * 10;
        }
    }

    /**
     * 主人公オブジェクトコンストラクタ
     */
    function Player(x, y) { ←2
        this.x = x;
        this.y = y;
        this.dx = 0;
        this.dy = 0;
        this.dir = 0;
        this.scrollCount = 0;

        this.update = function () {
            this.doScroll();
            if (this.scrollCount > 0) {
                return;
            }

            if (this.x == W - 2 && this.y == W - 2) {
                clearInterval(timer);
                status = GAMECLEAR;
                document.getElementById("bgm").pause();
                repaint();
            }

            this.dx = 0;
            this.dy = 0;
            var nx = 0, ny = 0;
            switch (keyCode) {
                case 37: nx = -1; ny = 0;
                    this.dir = 2;
                    break;
                case 38: nx = 0; ny = -1;
                    this.dir = 0;
```

```
                break;
            case 39: nx = +1; ny = 0;
                this.dir = 3;
                break;
            case 40: nx = 0; ny = +1;
                this.dir = 1;
                break;
        }
        if (maze[this.y + ny][this.x + nx] == 0) {
            this.dx = nx;
            this.dy = ny;
        }
    }

    this.paint = function (gc, x, y, w, h) {
        var img = document.getElementById("hero" + this.dir);
        gc.drawImage(img, x, y, w, h);
    }
}

/**
 * 敵オブジェクトコンストラクタ
 */
function Alien(x, y) {  ←3
    this.x = x;
    this.y = y;
    this.dx = 0;
    this.dy = 0;
    this.dir = 0;
    this.scrollCount = 0;

    this.update = function () {
        this.doScroll();

        // 衝突判定
        var diffX = Math.abs(player.getScrollX() - this.getScrollX());
        var diffY = Math.abs(player.getScrollY() - this.getScrollY());
        if (diffX <= 40 && diffY <= 40) {
            clearInterval(timer);
            status = GAMEOVER;
            document.getElementById("bgm").pause();
            repaint();
        }
```

```
        // 次の移動先
        var gapx = player.x - this.x;
        var gapy = player.y - this.y;
        switch (random(4)) {
            case 0:
                this.dx = gapx > 0 ? 1 : -1;
                this.dir = (this.dx == 37) ? 2 : 3;
                break;
            case 1:
                this.dy = gapy > 0 ? 1 : -1;
                this.dir = (this.dy == 38) ? 0 : 1;
                break;
            default:
                this.dx = 0;
                this.dy = 0;
                break;
        }
    }

    this.paint = function (gc, w, h) {
        var img = document.getElementById("alien" + this.dir);
        gc.drawImage(img, this.getScrollX(), this.getScrollY(), w, h);
    }
}

function random(v) {
    return Math.floor(Math.random() * v);
}

function init() {    ←4
    var maze = document.getElementById("maze");
    ctx = maze.getContext("2d");
    ctx.font = "bold 48px sans-serif";

    createMaze(W, H);
    player = new Player(1, 1);
    aliens = [new Alien(W - 2, 1), new Alien(1, W - 2)];
    repaint();
}

function go() {    ←5
    window.onkeydown = mykeydown;
    window.onkeyup = mykeyup;
```

```
    var maze = document.getElementById("maze");
    maze.onmousedown = mymousedown;
    maze.onmouseup = mykeyup;
    maze.oncontextmenu = function (e) { e.preventDefault(); };
    maze.addEventListener('touchstart', mymousedown);
    maze.addEventListener('touchend', mykeyup);

    timer = setInterval(tick, 45);
    document.getElementById("START").style.display = "none";
    document.getElementById("bgm").play();
}

/**
 * メインルーチン
 */
function tick() {    ←6
    player.update();
    for (var i = 0 ; i < aliens.length ; i++) {
        aliens[i].update();
    }
    repaint();
}

/**
 * 幅:w、高さ:hの迷路生成
 */
function createMaze(w, h) {        ←7
    for (var y = 0 ; y < h ; y++) {
        maze[y] = [];
        for (var x = 0 ; x < w ; x++) {        ←8
            maze[y][x] = (x == 0||x == w-1||y == 0||y == h-1) ? 1 : 0;
        }
    }

    for (var y = 2 ; y < h - 2 ; y += 2) {        ←9
        for (var x = 2 ; x < w - 2 ; x += 2) {
            maze[y][x] = 1;

            // 最上段は上下左右、それ以外は下左右
            var dir = random((y == 2) ? 4 : 3);
            var px = x, py = y;
            switch (dir) {
```

```
                case 0: py++; break;
                case 1: px--; break;
                case 2: px++; break;
                case 3: py--; break;
            }
            maze[py][px] = 1;
        }
    }
}

function drawCircle(x, y, r, color) {          ⬅10
    ctx.fillStyle = color;
    ctx.beginPath();
    ctx.arc(x, y, r, 0, Math.PI * 2);
    ctx.fill();
}

/**
 * 描画
 */
function repaint() {          ⬅11
    // 背景クリア
    ctx.fillStyle = "black";
    ctx.fillRect(0, 0, 900, 600);

    // クリップ領域設定
    ctx.save();
    ctx.beginPath();
    ctx.arc(300, 300, 300, 0, Math.PI * 2);
    ctx.clip();

    // 迷路描画
    ctx.fillStyle = "brown";
    ctx.translate(6 * 50, 6 * 50);
    ctx.translate(-1 * player.getScrollX(), -1 * player.getScrollY());
    for (var x = 0; x < W ; x++) {
        for (var y = 0; y < H ; y++) {
            if (maze[y][x] == 1) {
                ctx.fillRect(x * 50, y * 50, 50, 50);
            }
        }
    }
    for (var i = 0 ; i < aliens.length ; i++) {
        aliens[i].paint(ctx, 50, 50);
    }
```

```
    ctx.restore();

    // 地図描画
    ctx.fillStyle = "#eeeeee";
    ctx.fillRect(650, 0, 250, 600);

    ctx.save();
    ctx.translate(670, 300);
    ctx.fillStyle = "brown";
    for (var x = 0; x < W; x++) {
        for (var y = 0; y < H; y++) {
            if (maze[y][x] == 1) {
                ctx.fillRect(x * 7, y * 7, 7, 7);
            }
        }
    }
    drawCircle(player.x * 7 + 3, player.y * 7 + 3, 3, "red");
    for (var i = 0 ; i < aliens.length ; i++) {
        drawCircle(aliens[i].x * 7 + 3, aliens[i].y * 7 + 3, 3, "purple");
    }
    ctx.restore();

    // コントローラ描画
    ctx.drawImage(arrows, 670, 70, 200, 200);
    var ax = -100, ay = -100;
    switch (keyCode) {
        case 39: ax = 830; ay = 170; break;
        case 40: ax = 770; ay = 230; break;
        case 37: ax = 710; ay = 170; break;
        case 38: ax = 770; ay = 120; break;
    }
    drawCircle(ax, ay, 30, "yellow");

    // 主人公描画とメッセージ
    player.paint(ctx, 300, 300, 50, 50)
    ctx.fillStyle = "yellow";
    if (status == GAMEOVER) {
        ctx.fillText("GAME OVER", 150, 200);
    } else if (status == GAMECLEAR) {
        ctx.fillText("GAME CLEAR", 150, 200);
    }
}
```

```
        // キー&マウス押下のイベントハンドラ
        function mykeydown(e) {
            keyCode = e.keyCode;
        }
        function mykeyup(e) {                        ←12
            keyCode = 0;
        }
        function mymousedown(e) {             ←13
            var mouseX = !isNaN(e.offsetX) ? e.offsetX : e.touches[0].clientX;
            var mouseY = !isNaN(e.offsetY) ? e.offsetY : e.touches[0].clientY;
            if (670 < mouseX && mouseX < 870 && 70 < mouseY && mouseY < 270) {
                mouseX -= 770;
                mouseY -= 170;
                if (Math.abs(mouseX) > Math.abs(mouseY)) {
                    keyCode = mouseX < 0 ? 37 : 39;
                } else {
                    keyCode = mouseY < 0 ? 38 : 40;
                }
            }
        }
    </script>
</head>
<body onload="init()">
    <!-- Thanks to http://takao-suenobu.com/  & http://dova-s.jp/ -->
    <audio src="Emergency.mp3" id="bgm" loop="loop"></audio>
    <canvas id="maze" width="900" height="600"></canvas>
    <img id="START" src="start.png" onclick="go()"><br/>
    <img id="arrows" src="arrows.png" style="display:none" />

    <img id="hero0" src="hero0.png" style="display:none" />
    <img id="hero1" src="hero1.png" style="display:none" />
    <img id="hero2" src="hero2.png" style="display:none" />
    <img id="hero3" src="hero3.png" style="display:none" />
    <img id="alien0" src="alien0.png" style="display:none" />
    <img id="alien1" src="alien1.png" style="display:none" />
    <img id="alien2" src="alien2.png" style="display:none" />
    <img id="alien3" src="alien3.png" style="display:none" />
</body>
</html>
```

5-5-1 ソースコード解説

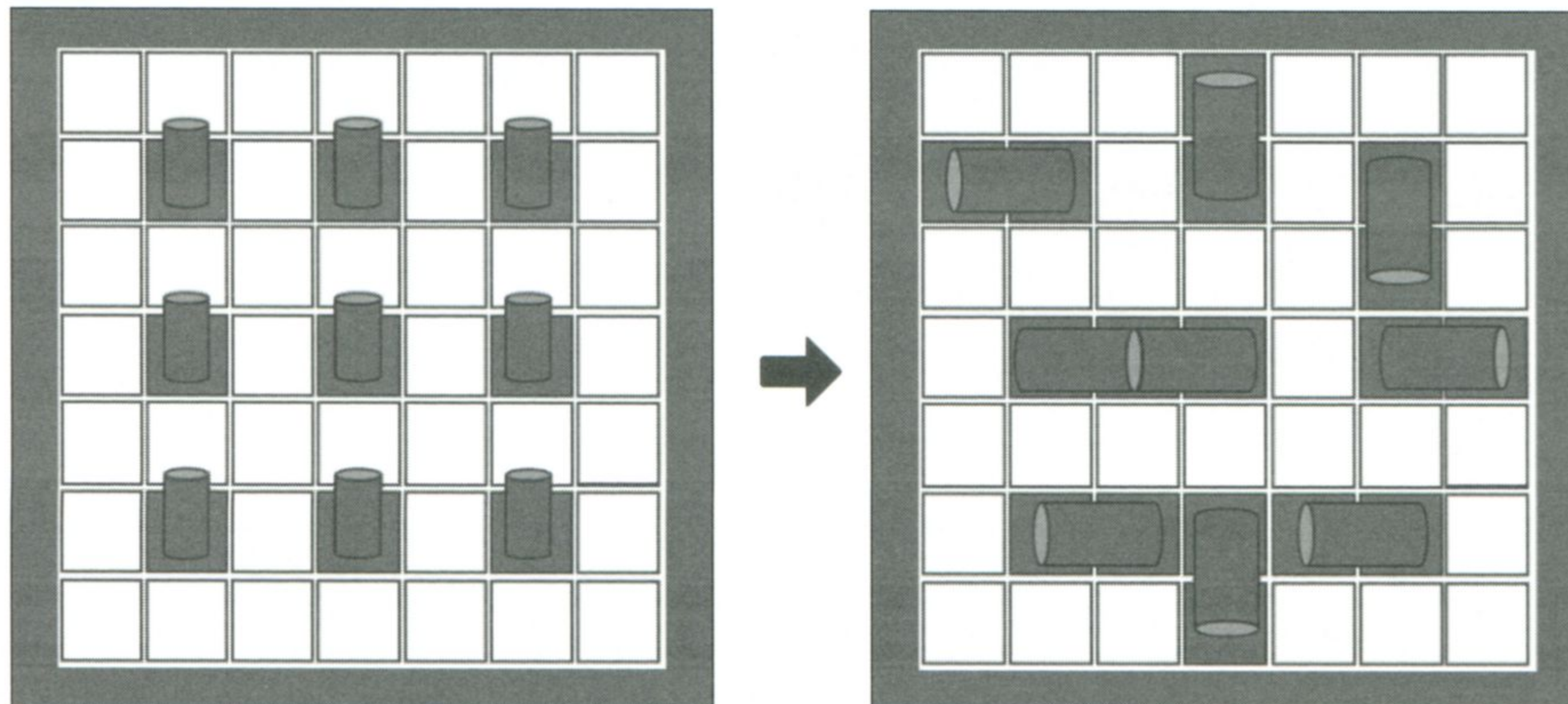
使用している広域変数は以下のとおりです。

使用している広域変数

変数	説明
ctx	グラフィックコンテキスト
keyCode	現在どのキーが押されているか
maze	迷路を保持する2元配列
player	主人公オブジェクト
aliens	敵オブジェクトを含む配列
timer	タイマー
status	ゲームの状態、クリア＝1、ゲームオーバー＝2
scroller	スクロール処理するオブジェクト、PlayerとAlienのプロトタイプ

▶棒倒し法による迷路作成

迷路を作成するアルゴリズムはたくさんありますが、今回はその中でも最もシンプルな棒倒し法を使用することにしました。簡単に説明すると、格子状に柱をたてて、その柱を4方向のどちらかへ倒していくことで迷路を作成するという方法です。乱数を使用するので実行するたびに異なる迷路が生成されます。その処理を行っているのが7のcreateMaze(w, h)関数です。アルゴリズムは非常にシンプルです。迷路データは2次元配列として格納し、壁を1通路を0としています。

棒倒し法による迷路作成

まず8のfor文で、上下左右の壁を1にします。

```
for (var y = 0 ; y < h ; y++) {
    maze[y] = [];
    for (var x = 0 ; x < w ; x++) {
        maze[y][x] = (x == 0||x == w-1||y == 0||y == h-1) ? 1 : 0;
    }
}
```

次に9のfor文で、以下のようにひとつ置きに柱を立てて行きます。前ページの図左の状態です。

```
for (var y = 2 ; y < h - 2 ; y += 2) {
    for (var x = 2 ; x < w - 2 ; x += 2) {
        maze[y][x] = 1;
```

あとは、最上段の柱（「y == 2」のとき）は上下左右4方向のどちらか、それ以外は下・左・右の3方向のどちらかに柱を倒します。柱を倒した場所は壁にします。ほかの柱が倒れているところには別の柱を倒さないようにするとより複雑な迷路が生成されます。

▶スムーズスクロール

迷路は碁盤目状のデータとして管理しています。上下左右に移動する際に隣のマスにジャンプして移動する実装だと、動きがカクカクしてぎこちないものになってしまいます。そこで、隣のマスに動くまでを5等分して、徐々に移動させることで、スムーズに動くようにしました。

主人公はオブジェクトとして実装していますが、座標に関連するプロパティは以下のとおりです。

主人公のプロパティ

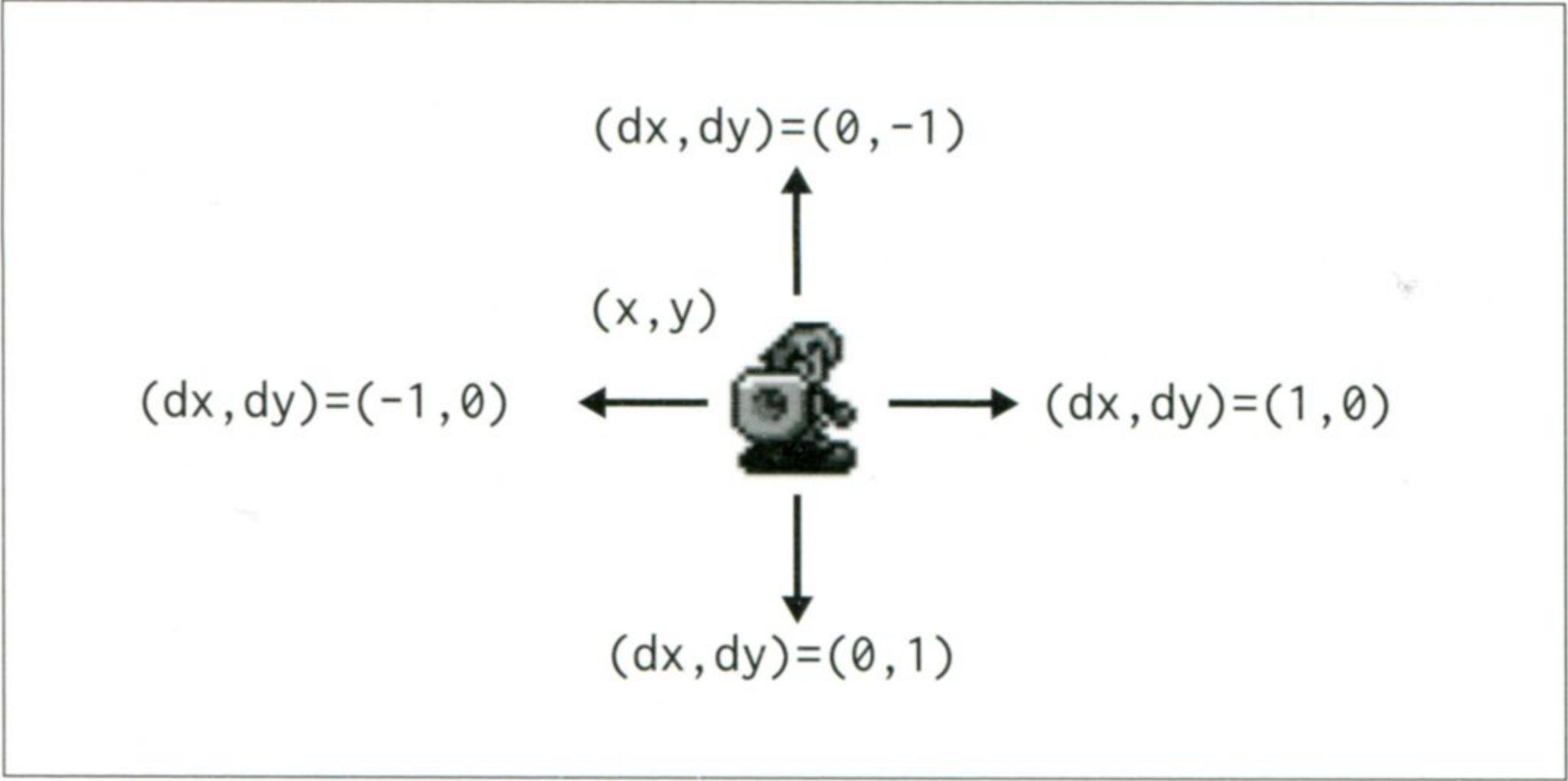
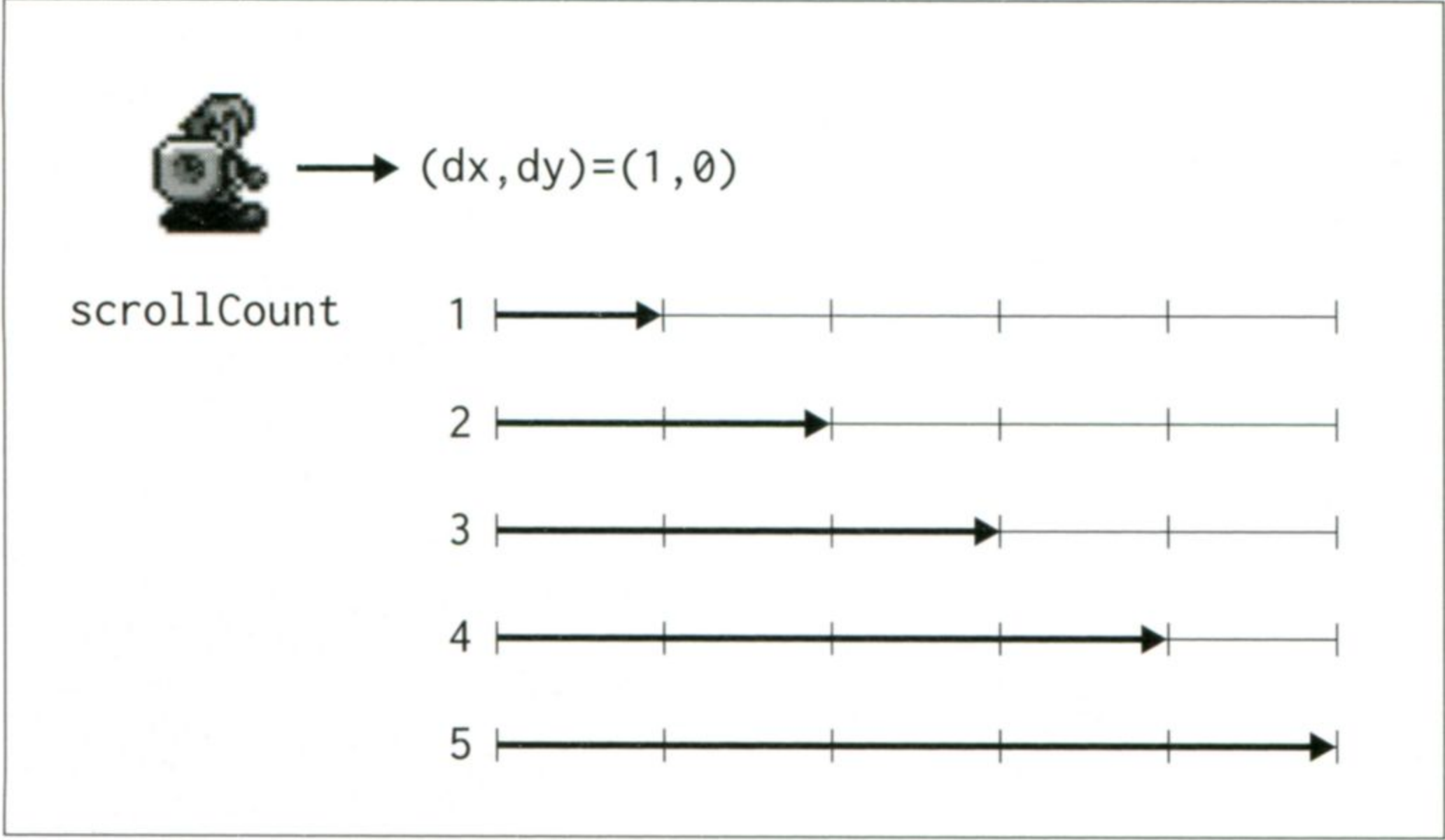
プロパティ	説明
(x, y)	現在のオブジェクトの座標
(dx, dy)	オブジェクトの移動方向
scrollCount	5等分したうちの何個めかを数えるカウンタ
dir	上下左右の画像の向き

(dx, dy)による移動方向と、右に移動する際の様子を以下に示します。

移動方向

(dx,dy)=(0,-1)
(x,y)
(dx,dy)=(-1,0)
(dx,dy)=(1,0)
(dx,dy)=(0,1)

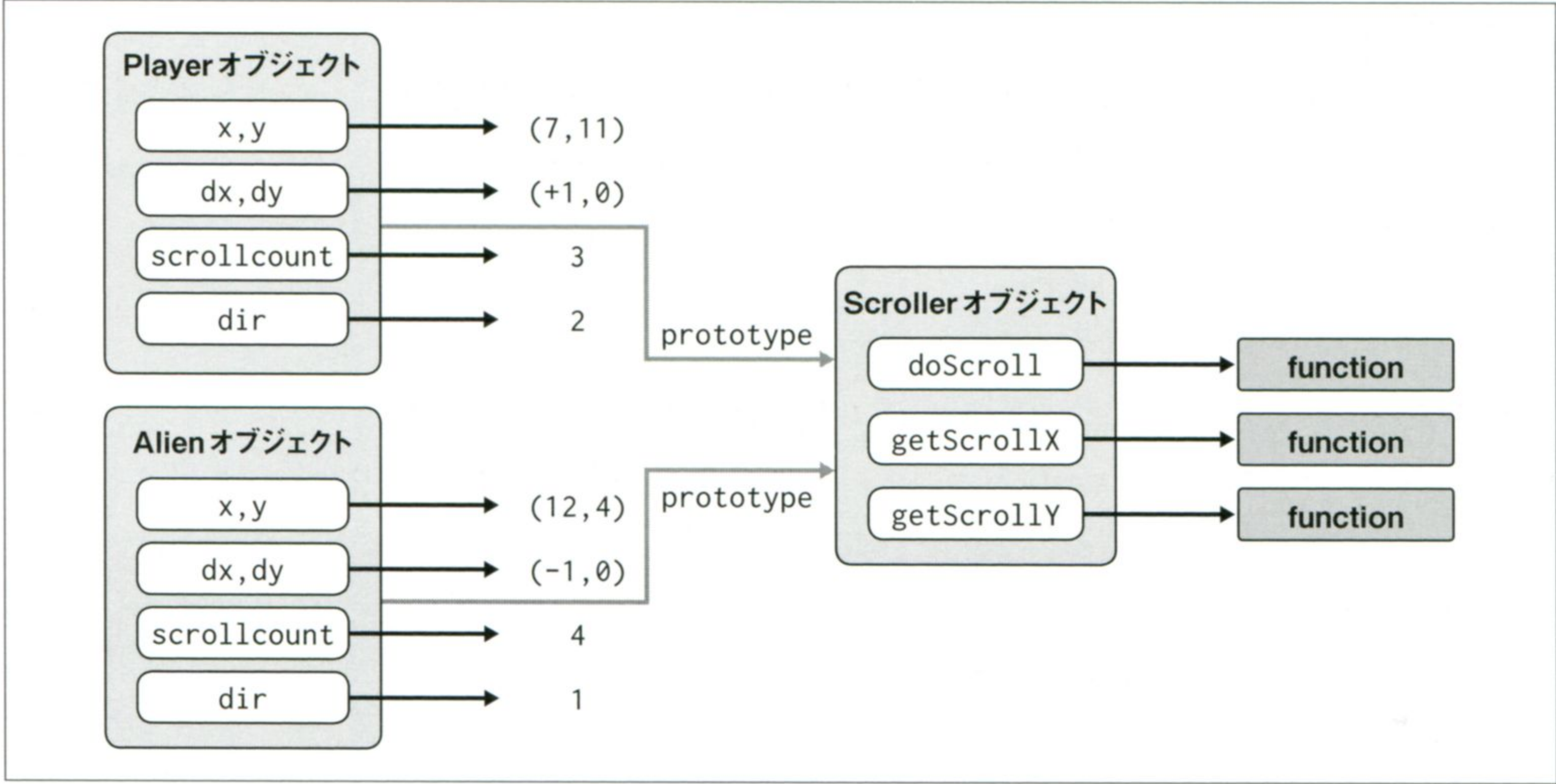
右に移動する様子

(dx,dy)=(1,0)
scrollCount
1
2
3
4
5

このゲームで移動するのは主人公Playerだけでなく、敵Alienのオブジェクトも同様に移動します。そこで、共通する処理をScrollerというオブジェクトにまとめ、prototypeとして参照することにしました。その様子を以下に示します。

Playerオブジェクト、AlienのオブジェクトはScrollerオブジェクトをprototypeとして参照する

Playerオブジェクト
x,y → (7,11)
dx,dy → (+1,0)
scrollcount → 3
dir → 2
prototype

Alienオブジェクト
x,y → (12,4)
dx,dy → (-1,0)
scrollcount → 4
dir → 1
prototype

Scrollerオブジェクト
doScroll → function
getScrollX → function
getScrollY → function

▶1 Scrollerオブジェクト

Scrollerオブジェクトの実装は以下のとおりです。

```
function Scroller() {

    this.doScroll = function () {        ←Ⓐ
        if (this.dx == 0 && this.dy == 0) {
            return;
        }

        if (++this.scrollCount >= 5) {
            this.x = this.x + this.dx;
            this.y = this.y + this.dy;
            this.dx = 0;
            this.dy = 0;
            this.scrollCount = 0;
        }
    }

    this.getScrollX = function () {      ←Ⓑ
        return this.x * 50 + this.dx * this.scrollCount * 10;
    }

    this.getScrollY = function () {      ←Ⓒ
        return this.y * 50 + this.dy * this.scrollCount * 10;
    }
}
```

ⒶdoScrollメソッド

スクロール用カウンタscrollCountを増加させます。(dx, dy)がともに0のときは何もせずにreturnします。その値が5以上になった場合、一コマ移動させるので、自分の座標(x, y)に(dx, dy)を加えて、カウンタと(dx, dy)を0で初期化します。

ⒷgetScrollXメソッド

現在のx座標を返します。1マス50ピクセル分なので座標は「this.x * 50」となります。さらにカウンタが5で1マス、すなわちカウンタ1で10ピクセル分の移動になるため、その分を加算しています。

ⒸgetScrollYメソッド

y座標の値を返します。実装はx座標と同じです。

▶2 Playerオブジェクト

Playerオブジェクトの実装は以下のとおりです。

```
function Player(x, y) {
    this.x = x;
    this.y = y;
    this.dx = 0;
    this.dy = 0;
    this.dir = 0;
    this.scrollCount = 0;

    this.update = function () {    ←Ⓐ
        this.doScroll();    ←Ⓑ
        if (this.scrollCount > 0) {
            return;
        }

        if (this.x == W - 2 && this.y == W - 2) {   ←Ⓒ
            clearInterval(timer);
            status = GAMECLEAR;
            document.getElementById("bgm").pause();
            repaint();
        }

        this.dx = 0;
        this.dy = 0;
        var nx = 0, ny = 0;
        switch (keyCode) {  ←Ⓓ
            case 37: nx = -1; ny = 0;
                this.dir = 2;
                break;
            case 38: nx = 0; ny = -1;
                this.dir = 0;
                break;
            case 39: nx = +1; ny = 0;
                this.dir = 3;
                break;
            case 40: nx = 0; ny = +1;
                this.dir = 1;
                break;
        }
        if (maze[this.y + ny][this.x + nx] == 0) {  ←Ⓔ
            this.dx = nx;
```

```
            this.dy = ny;
        }
    }

    this.paint = function (gc, x, y, w, h) {    ←Ⓕ
        var img = document.getElementById("hero" + this.dir);
        gc.drawImage(img, x, y, w, h);
    }
}
```

Ⓐupdateメソッド

Ⓑのthis.doScroll()でscrollbarオブジェクトのカウンタscrollCountを増やします。この値が0より大きい場合、スクロール中なので以降の処理は行わずreturnします。

Ⓒ「(this.x == W - 2 && this.y == W - 2)」のときは、迷路右下に到着したのでゴールとします。

Ⓓでは、keyCodeの値に応じて(nx, ny)とdirの値を更新します。

Ⓔの「(maze[this.y + ny][this.x + nx] == 0)」が成立する場合は移動先の場所が通路なので、(dx, dy)を更新します。

Ⓕpaintメソッド

主人公のimgオブジェクトを取得し、座標(x, y)に大きさ(w, h)で描画します。

▶3 Alienオブジェクト

Alienオブジェクトは敵のキャラクタです。

```
function Alien(x, y) {
    this.x = x;
    this.y = y;
    this.dx = 0;
    this.dy = 0;
    this.dir = 0;
    this.scrollCount = 0;

    this.update = function () { ←Ⓐ
        this.doScroll();    ←Ⓑ

        // 衝突判定
        var diffX = Math.abs(player.getScrollX() - this.getScrollX());  ←Ⓒ
        var diffY = Math.abs(player.getScrollY() - this.getScrollY());
```

```
        if (diffX <= 40 && diffY <= 40) {{  ←Ⓓ
            clearInterval(timer);
            status = GAMEOVER;
            document.getElementById("bgm").pause();
            repaint();
        }

        // 次の移動先   ←Ⓔ
        var gapx = player.x - this.x;
        var gapy = player.y - this.y;
        switch (random(4)) {
            case 0:
                this.dx = gapx > 0 ? 1 : -1;
                this.dir = (this.dx == -1) ? 2 : 3;
                break;
            case 1:
                this.dy = gapy > 0 ? 1 : -1;
                this.dir = (this.dy == -1) ? 0 : 1;
                break;
            default:
                this.dx = 0;
                this.dy = 0;
                break;
        }
    }

    this.paint = function (gc, w, h) {
        var img = document.getElementById("alien" + this.dir);
        gc.drawImage(img, this.getScrollX(), this.getScrollY(), w, h);
    }
}
```

Ⓐのupdateメソッドでは、まずⒷのthis.doScroll()で、scrollbarオブジェクトのカウンタscrollCountを増やします。

その後、主人公キャラクタとの衝突判定を行います。

ⒸのMath.abs()は絶対値を求めるメソッドです。x方向の差分とy方向の差分がともに40以下になったときⒹに衝突として、ゲームオーバーの処理を行います。

Ⓔの移動先では、徐々に主人公に近づくようAlienの移動先を求めています。gapxは主人公のx座標から敵のx座標を引いた値ですが、この値が正の場合、主人公が右にいることになるので、移動方向dxは1としています。ただ、毎回確実に近づくと難易度が高くなりすぎるので乱数で調整しています。

▶4 init()

canvasのコンテキストを設定し、createMaze(W, H)を呼び出して迷路データを作成しています。そして、主人公、Alienとオブジェクトを作成しています。この段階ではゲームはまだ開始していないことに注意してください。ゲームはSTARTボタンが押されてgo()が呼び出されてから始まります。

このように、STARTボタンの押下ではじめてゲームが開始されますが、これには理由があります。HTML/JavaScriptで実装されているゲームの多くはスマホでも動作します。従量課金契約の場合、大量のデータを受信すると高額な通信料が課される可能性があります。「HTMLのページを閲覧しただけなのに、知らない間に映像や音声のダウンロードが行われ高額な料金が請求された」という状況が発生しないよう、ユーザーの操作に応じた場合でしか音声が再生されないようになっている携帯端末が存在します。

このような端末の場合、onloadのように自動的に呼び出されるコールバック関数の中でBGM再生用のコードを記述しても音楽の再生が行われません。スマホでもできるだけPCと同じ挙動になるようにしたかったので、明示的にユーザーがSTARTボタンを押下したときにゲームが開始される仕様にしました。

▶5 go()

キー、マウス、タッチ用のイベントハンドラを登録しています。以下の行はタッチの長押しによってコンテキストメニューが表示される挙動を防止するためのものです。

```
maze.oncontextmenu = function (e) { e.preventDefault(); };
```

その後、setIntervalでメインループのtickを開始し、STARTボタンを非表示にして、BGMの再生を開始しています。

```
timer = setInterval(tick, 45);
document.getElementById("START").style.display = "none";
document.getElementById("bgm").play();
```

ちなみに、イベントハンドラをinit()でなく、go()の中で登録しているのには理由があります。もしinit()の中でイベントハンドラを登録してしまうと、STARTボタン以外の場所がクリックされると、ゲーム開始前にも関わらず、イベントハンドラが実行されてしまいます。これは意図する挙動でなはいため、ゲーム開始直前のgo()で登録しています。

▶6 tick()

ゲームのメインループです。PlayerとAlienのupdate()メソッドを呼び出し、repaint()で再描画をしています。

▶10 drawCircle(x, y, r, color)

円をcanvasに描画します。このゲームでは円を描く処理が色々な場所に散見されたので、関数としてまとめました。

▶11 repaint()

まず背景を黒で塗りつぶします。ctx.save()でコンテキストを保存します。ctx.arc(…)で円を描き、ctx.clip()を呼ぶことでクリップ領域を設定しています。こうするとctx.restore()が呼ばれるまで、描画処理はこの円の中だけに限定されます。

このゲームでは常に主人公は中心に描画されます。そこで、迷路を主人公と反対側に動かすことでスクロール効果を演出しています。その座標系を移動する処理をctx.translate()で行っています。あとは二重ループで壁の部分を茶色で塗りつぶし、敵キャラクタを描画しています。

「// 地図描画」以降で、画面右にある小さな地図を描画しています。これも同じく二重ループを使い、小さな地図を描画しています。自分を赤色で、敵を紫で描画しています。最後に、主人公、必要に応じてメッセージを描画しています。

▶12 mykeydown(e)、mykeyup(e)

押下されているキーのコードを広域変数keyCodeに格納します。キーが離れた時は0を代入します。

▶13 mymousedown(e)

マウスクリックやタッチ時のコールバック関数です。ひとつのコールバック関数をマウスとタッチで共用するため、以下のコードで座標位置を取得しています。

```
var mouseX = !isNaN(e.offsetX) ? e.offsetX : e.touches[0].clientX;
var mouseY = !isNaN(e.offsetY) ? e.offsetY : e.touches[0].clientY;
```

isNaN()は数値でないか否かを返す関数です。マウスがクリックされている場合は、e.offsetXに数値が格納されているので、「!isNaN(e.offsetX)」がtrueになり、e.offsetXの値がmouseXに設定されます。数値でない場合はe.touches[0].clientXでタッチ座標を設定しています。

実は、最初のバージョンでは以下のようなコードを使用していました。

```
var mouseX = e.offsetX || e.touches[0].clientX;
```

e.offsetXがtrueと判断されればその値を、そうでない場合はe.touches[0].clientXを使用します。何が問題

か分かりますか？　マウスの座標が0のとき、すなわちoffsetXの値が0のときを考えてみてください。0は条件式としてはfalseとなるので、||の右側の式が評価されます。マウスで操作しているのにタッチのプロパティが参照されてしまうのです。

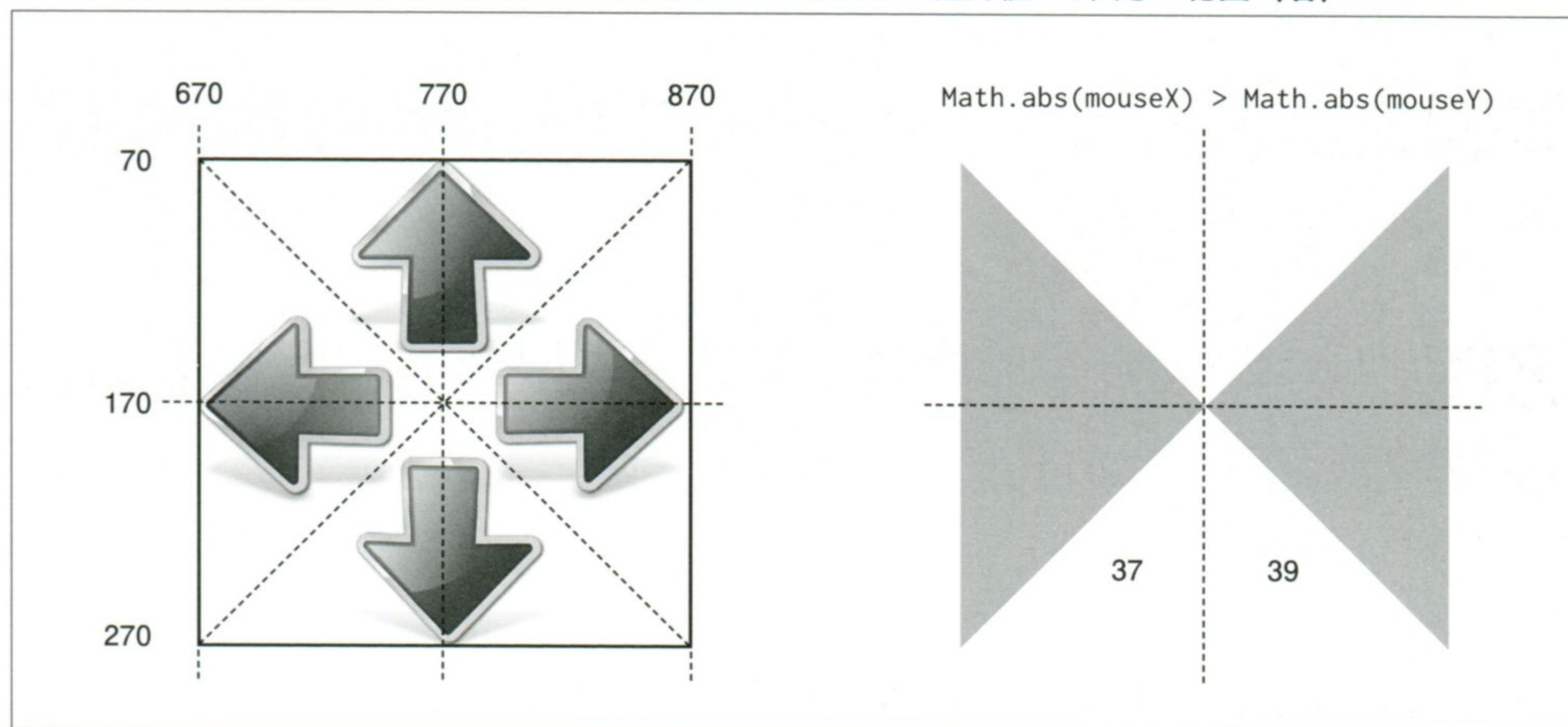
座標値が得られたら、どのボタンの上で押されたのか判定する処理へ進みます。まず、

```
if (670 < mouseX && mouseX < 870 && 70 < mouseY && mouseY < 270) {
```

で座標値が上下左右ボタンの領域に含まれているか判定します。正確に矢印ボタンの上がクリックされているか判定する必要はありません。対角線を引いて大雑把に調べれば十分です。

左右ボタンを判定する場合は、「(Math.abs(mouseX) > Math.abs(mouseY))」を評価します。Math.abs()は絶対値を返す関数です。mouseXの絶対値がmouseYの絶対値より大きい範囲は下右図のようになります。

ここまでわかれば、mouseXの値が原点より右にあるか左にあるか判定することで、左右どちらのボタンか判断することが可能になります。上下ボタンも同じです。

上下左右ボタンの領域（左）mouseXの絶対値がmouseYの絶対値より大きい範囲（右）

演習　迷路のデザインを変えてみよう

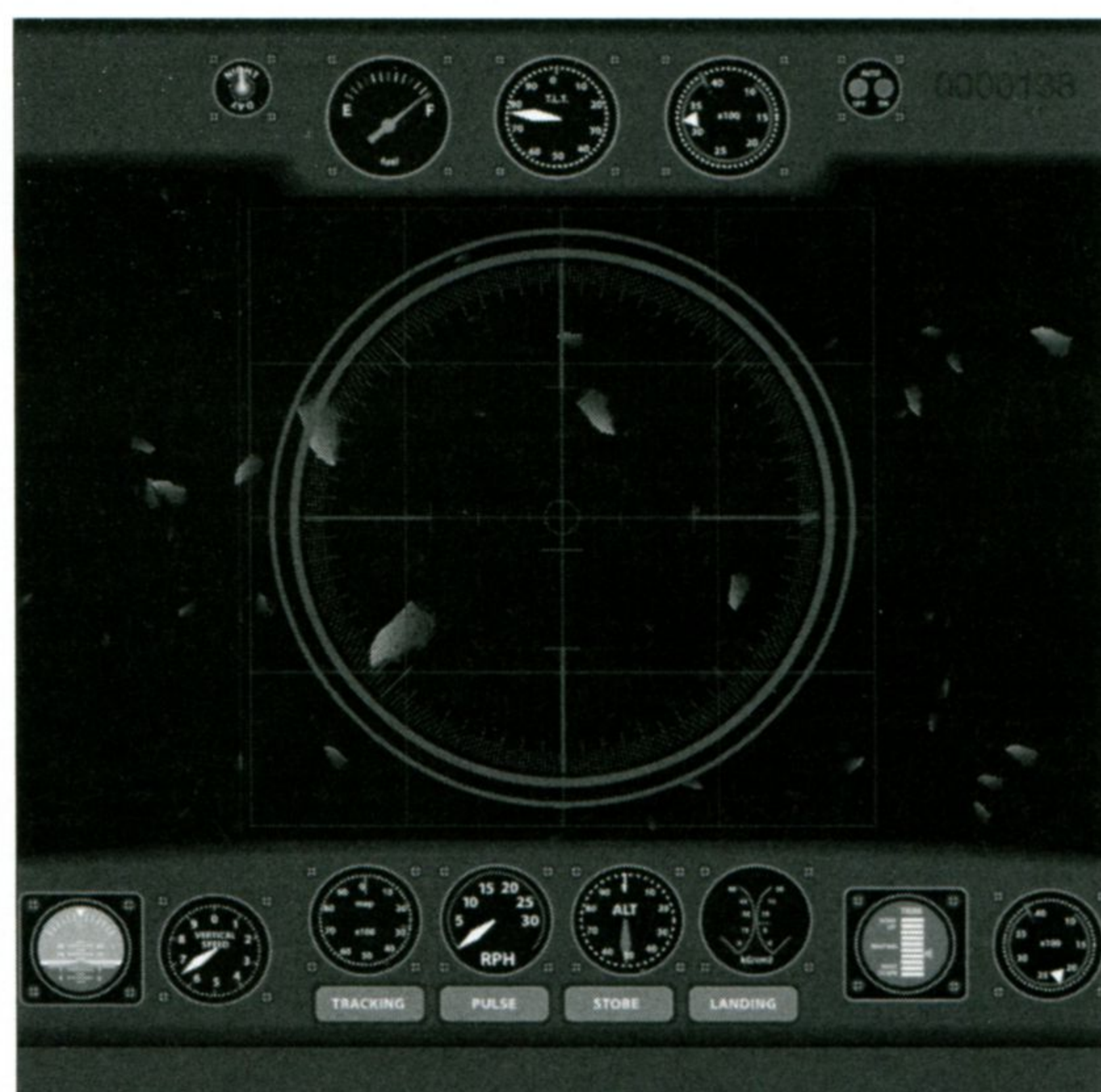
クリップ領域を円ではなく、ほかの形にしてみましょう。また迷路サイズを変えてみたり、迷路作成のアルゴリズムをより複雑なものに変えてみたりといろいろ手を加えてみましょう。

5-6 Saturn Voyager

本格的な3Dゲームをつくるのは大変です。3Dモデルを作って、テクスチャを貼り付けて等々、さまざまな処理が必要です。しかし、「なんちゃって3Dゲーム」であればそれほど面倒ではありません。簡単な割には実際に3D空間を移動しているような錯覚を覚えると思います。

このゲームで学ぶこと

- Canvasの座標軸変換になれる（画像の回転）
- 疑似3Dモデルに親しむ

Let's write codes !

```
<!DOCTYPE html>
<html>
<head>
    <title>SaturnVoyager</title>
    <META charset="UTF-8">
    <style>
        #space {
            width: 800px; height: 800px;
            touch-action: none;
        }
        #START {
            position: absolute;
            left: 200px;
            top: 200px;
        }
    </style>
    <script>
        "use strict";
        var stars = [], keymap = [];
        var ctx, ship, score = 0, speed = 25, timer = NaN;

        function Ship(x, y) {    ←1
            this.x = x;          ←2
            this.y = y;
            this.keydown = function (e) {
                keymap[e.keyCode] = true;
            }                                    ←3
            this.keyup = function (e) {
                keymap[e.keyCode] = false;
            }
            this.move = function () {
                if (keymap[37]) {           // 左
                    this.x -= 30;
                } else if (keymap[39]) {    // 右
                    this.x += 30;
                }

                if (keymap[38]) {           // 上          ←4
                    this.y -= 30;
                } else if (keymap[40]) {    // 下
                    this.y += 30;
                }
                this.x = Math.max(-800, Math.min(800, this.x));
```

```
        this.y = Math.max(-800, Math.min(800, this.y));
    }
}

function random(v) {     ←5
    return Math.floor(Math.random() * v);
}

function init() {        ←6
    for (var i = 0 ; i < 200 ; i++) {
        stars.push({
            x: random(800 * 4) - 1600,
            y: random(800 * 4) - 1600,
            z: random(4095),
            r: random(360),
            w: random(10) - 5
        });
    }

    ship = new Ship(200, 200);
    onkeydown = ship.keydown;
    onkeyup = ship.keyup;
                                                        ←7
    var space = document.getElementById("space");
    ctx = space.getContext("2d");
    ctx.font = "20pt Arial";
    repaint();
}

function go() {      ←8
    var space = document.getElementById("space");
    space.onmousedown = mymousedown;
    space.onmouseup = mymouseup;
    space.oncontextmenu = function (e) { e.preventDefault(); };
    space.addEventListener('touchstart', mymousedown);
    space.addEventListener('touchend', mymouseup);

    document.body.addEventListener('touchmove', function (event) {
        event.preventDefault();                                     ←9
    }, false);
    document.getElementById("START").style.display = "none";
    document.getElementById("bgm").play();                          ←10
    timer = setInterval(tick, 50);
}
```

```
function mymousedown(e) {        ←11
    var mouseX = (!isNaN(e.offsetX) ? e.offsetX : e.touches[0].clientX) - 400;
    var mouseY = (!isNaN(e.offsetY) ? e.offsetY : e.touches[0].clientY) - 400;
    if (Math.abs(mouseX) > Math.abs(mouseY)) {
        keymap[mouseX > 0 ? 37 : 39] = true;
    } else {
        keymap[mouseY > 0 ? 38 : 40] = true;
    }
}

function mymouseup(e) {
    keymap = [];
}

function tick() {        ←12
    for (var i = 0 ; i < 200 ; i++) {
        var star = stars[i];
        star.z -= speed;        ←13
        star.r += star.w;        ←14
        if (star.z < 64) {
            if (Math.abs(star.x - ship.x) < 50 &&
                Math.abs(star.y - ship.y) < 50) {
                // 衝突→ゲームオーバー
                clearInterval(timer);
                timer = NaN;
                document.getElementById("bgm").pause();        ←15
                break;
            }
            // 通過→奥へ再配置
            star.x = random(800 * 4) - 1600;
            star.y = random(800 * 4) - 1600;
            star.z = 4095;
        }
    }
    if (score++ % 10 == 0) {
        speed ++;
    }        ←16
    ship.move();
    repaint();
}

function repaint() {        ←17
    ctx.fillStyle = "black";
    ctx.fillRect(0, 0, 800, 800);
```

```
        stars.sort(function (a, b) {
            return b.z - a.z;                    ←18
        });

        // 隕石の描画
        for (var i = 0 ; i < 200 ; i++) {
            var star = stars[i];
            var z = star.z;
            var x = ((star.x - ship.x) << 9) / z + 400;
            var y = ((star.y - ship.y) << 9) / z + 400;
            var size = (50 << 9) / z;                                        ←19
            ctx.save();
            ctx.translate(x, y);
            ctx.globalAlpha = 1- (z / 4096);
            ctx.rotate(star.r * Math.PI / 180);
            ctx.drawImage(rockImg, -size / 2, -size / 2, size, size);
            ctx.restore();
        }

        // スコア
        ctx.drawImage(scope, 0, 0, 800, 800);
        ctx.fillStyle = "green";
        ctx.fillText(('0000000' + score).slice(-7), 670, 60);    ←20
        if (isNaN(timer)) {
            ctx.fillText("GAME OVER", 315, 350);
        }
    }
</script>
</head>
<body onload="init()">
    <!-- Thanks to http://takao-suenobu.com/  & http://dova-s.jp/ -->
    <audio src="Escape.mp3" id="bgm" loop="loop"></audio>
    <canvas id="space" width="800" height="800"></canvas>
    <img id="START" src="start.png" onclick="go()"><br/>     ←21
    <img id="rockImg" src="rock.png" style="display:none" />
    <img id="scope" src="scope.png" style="display:none" />
</body>
</html>
```

5-6-1 ソースコード解説

使用している広域変数は以下のとおりです。

使用している広域変数

変数	説明
stars	隕石の場所を保持する配列
keymap	現在どのキーが押されているか保持する配列
ctx	グラフィックコンテキスト
ship	自機のオブジェクト
score	現在の点数
speed	スピード
timer	タイマー

▶1 Ship(x, y)

自機オブジェクトを作るためのコンストラクタです。引数xとyは自機の座標です。

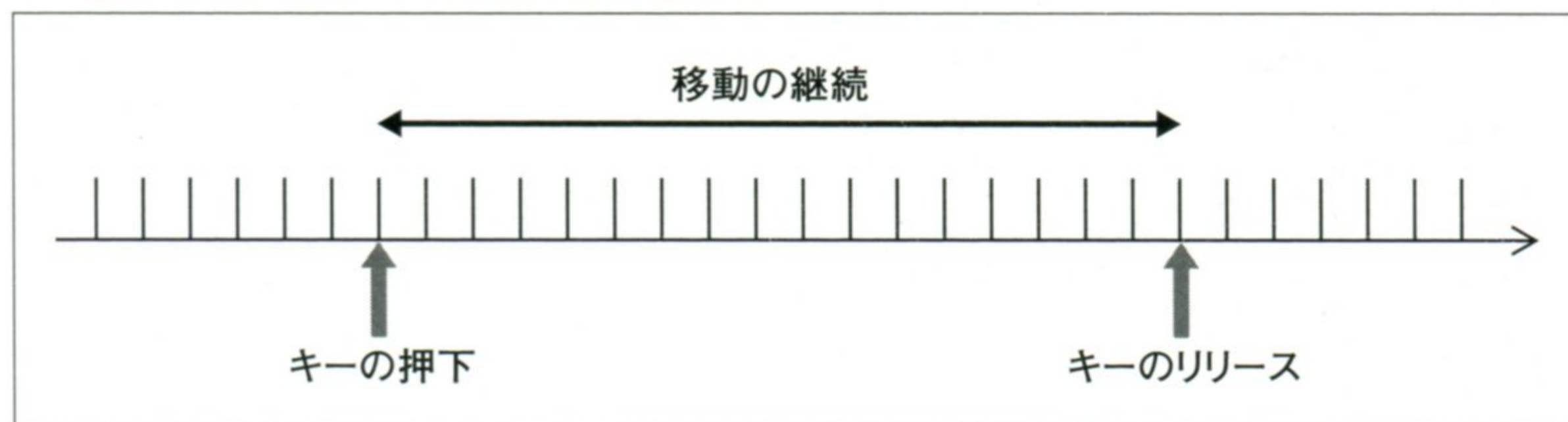
メソッドとプロパティは以下のとおりです。

2 x, y

XとYの座標値を保持するプロパティです。それぞれ-800から800の範囲をとります。

3 keydown, keyup

キーの押下状態を保持するためのメソッドです。コードをみるとわかりますが、どのキーが押されたかを配列keymapに記憶しているだけです。キーが押されたので、自機を動かす処理を書きたい衝動に駆られるかもしれません。なぜ、このメソッドの中で自機を移動する処理を行わないか、その理由を考えてみましょう。

キーを押している間は継続して移動させる

このようなリアルタイムゲームの場合、キーが押している間はキャラクタを継続して移動させることが普通です。リアルタイムゲームでは定期的にメインループが実行されます。メインループの中で都度キーの押下状態をチェックして、押下されているときに移動処理を行うと、キー押下の間だけ移動するという自然な動きが実現できるのです。keydownに移動処理を書いてしまうと「キーを押すたびに、ちょっとだけ移動する」という状態になってしまうでしょう。

4 move

キーの押下状態を見て自機を移動します。左右移動の処理を見てみましょう。

```
if (keymap[37]) {           // 左
    this.x -= 30;
} else if (keymap[39]) {    // 右
    this.x += 30;
}

if (keymap[38]) {           // 上
    this.y -= 30;
} else if (keymap[40]) {    // 下
    this.y += 30;
}
```

keydown, keyupでみたように、左キーが押下されているときにはkeymap[37]がtrueになります。そこで自機のx座標を30減らします。この値を調整すると自機が動く速さを制御できます。右、上、下、すべて同様です。

左右と上下ではif文が分かれています。すべてのケースをelse ifでつないでいないことに注意してください。これは、左と上が同時に押されたときは左上へ、といった斜め方向への移動を可能にするためです。

最後に以下のようにMath.maxとMath.minを使って変数の範囲を制限しています。

```
this.x = Math.max(-800, Math.min(800, this.x));
```

このような式は通常の計算式と同様に内側から見ていきます。まず、「Math.min(800, this.x)」でxと800の小さいほうを取得します。これで上限が800に制限されます。

次に「Math.max(−800,…)」で、その戻り値と−800を比べて大きい値を取得します。このように記述することで、if文を使用することなく1行で変数の範囲を制限することができます。ちょっと覚えておくと便利な小技です。

▶5 random(v)

vまでの範囲でランダムな整数値を返します。

▶6 init()

ゲームの初期化を行います。隕石を200個生成し、配列starsに格納しています。隕石オブジェクトの生成は次のコードで行っています。

```
stars.push({
    x: random(800 * 4) - 1600,
    y: random(800 * 4) - 1600,
    z: random(4095),
    r: random(360),
    w: random(10) - 5
});
```

波括弧「{}」でオブジェクトを作成し、「プロパティ名：プロパティ値;」で、プロパティやメソッドを指定できたことを覚えていますか？　忘れている人は「3-7-2 JavaScriptでのオブジェクトの定義方法」（P.102）を読み直してください。x, y , zは隕石の初期座標値です。

隕石は、x,yともに - 1600から1600までの範囲、zは0から4095までの範囲に配置されます。rは隕石の初期角度、wは隕石の回転するスピードです。すべてrandomで求めています。ちなみに自機はz=0の平面上を移動し、その座標は(x,y)となります。

7で自機のオブジェクトを作成し、キー押下用のイベントハンドラを設定しています。

```
ship = new Ship(200, 200);
onkeydown = ship.keydown;
onkeyup = ship.keyup;
```

あとはcanvasのコンテキストにフォントを設定し、repaint()で再描画を行っています。

▶ 8 go()

21のSTARTボタンの押下で呼び出される関数です。実際にゲームを開始する関数です。canvasにマウスやタッチのイベントハンドラを設定しています。

タッチで指を動かしたときに、ブラウザによっては、ゲームの操作ではなく、ゲーム画面を移動する操作と解釈するものがあったため以下のコードを追加しました9。

```
document.body.addEventListener('touchmove', function (event) {
    event.preventDefault();
}, false);
```

タッチは割と新しい技術要素であることもあり、ブラウザによって挙動が異なることも少なくありません。しばらくは個々のブラウザに対処するためのコードを書く必要があるでしょう。

後は、10でSTARTボタンを非表示にし、BGMの再生を開始し、メインループtickを開始しています。

```
document.getElementById("START").style.display = "none";
document.getElementById("bgm").play();
timer = setInterval(tick, 50);
```

▶11 mymousedown(e)

マウスの押下される場所に応じて上下左右キーが押下された場合と同じ処理を行っています。内容はDungeonのmymousedown(e)とほとんど同じです。

▶12 tick()

ゲームの心臓部ともいえるメインループです。

まず13の

```
star.z -= speed;
```

で隕石のz方向の値をspeed分減らしています。すなわち、自機の方へ隕石を近づけています。次に14の、

```
star.r += star.w;
```

で、隕石を回転させています。

次の15が衝突判定を行っている箇所です。

```
if (star.z < 64) {
    if (Math.abs(star.x - ship.x) < 50 &&
        Math.abs(star.y - ship.y) < 50) {
        // 衝突→ゲームオーバー
        clearInterval(timer);
        timer = NaN;
        document.getElementById("bgm").pause();
        break;
    }
    // 通過→奥へ再配置
    star.x = random(800 * 4) - 1600;
    star.y = random(800 * 4) - 1600;
    star.z = 4095;
}
```

「star.z < 64」で隕石がほぼ自分と同じ平面に到達したか判定しています。その場合、さらにx座標とy座標が至近であれば衝突したことになります。その判定をしているのが、以下の条件式です。

```
(Math.abs(star.x - ship.x) < 50 && Math.abs(star.y - ship.y) < 50)
```

Math.abs()は引数の絶対値を求めるメソッドです。つまり隕石と自機のx, y座標の差がともに50未満のときを衝突とみなしています。衝突した場合はタイマーとBGMを止めてゲームオーバーとしています。衝突しなかった場合は、自機の動く平面を通り過ぎたことになるので、以下のように、乱数を使って一番奥へ隕石を配置しています。

```
// 通過→奥へ再配置
star.x = random(800 * 4) - 1600;
star.y = random(800 * 4) - 1600;
star.z = 4095;
```

メインループでの残りの処理16は以下のとおりシンプルです。

```
if (score++ % 10 == 0) {
    speed ++;
}
ship.move();
repaint();
```

scoreを増やして10の倍数になったらspeedを増やします。これによりゲームを継続すると徐々にスピードが増加します。あとはship.move()で自機を移動させ、repaint()で画面を再描画します。

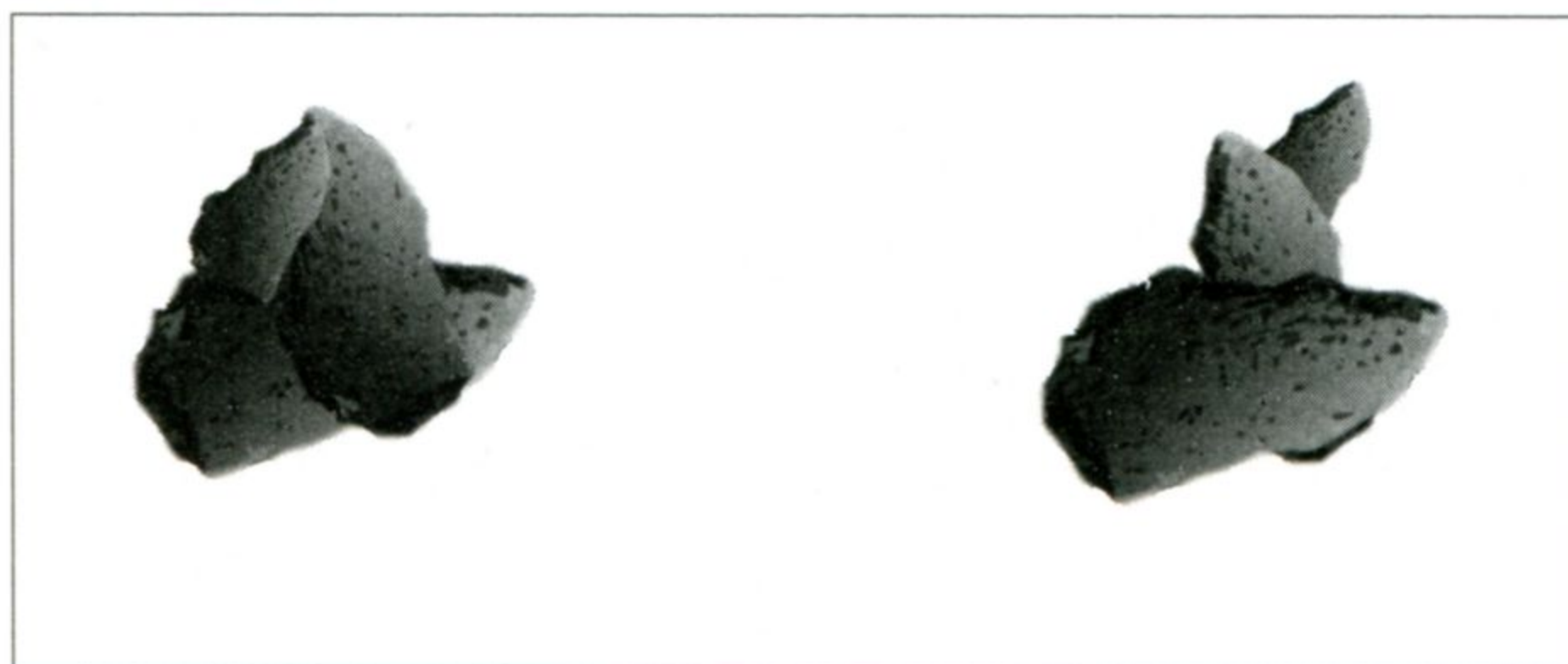
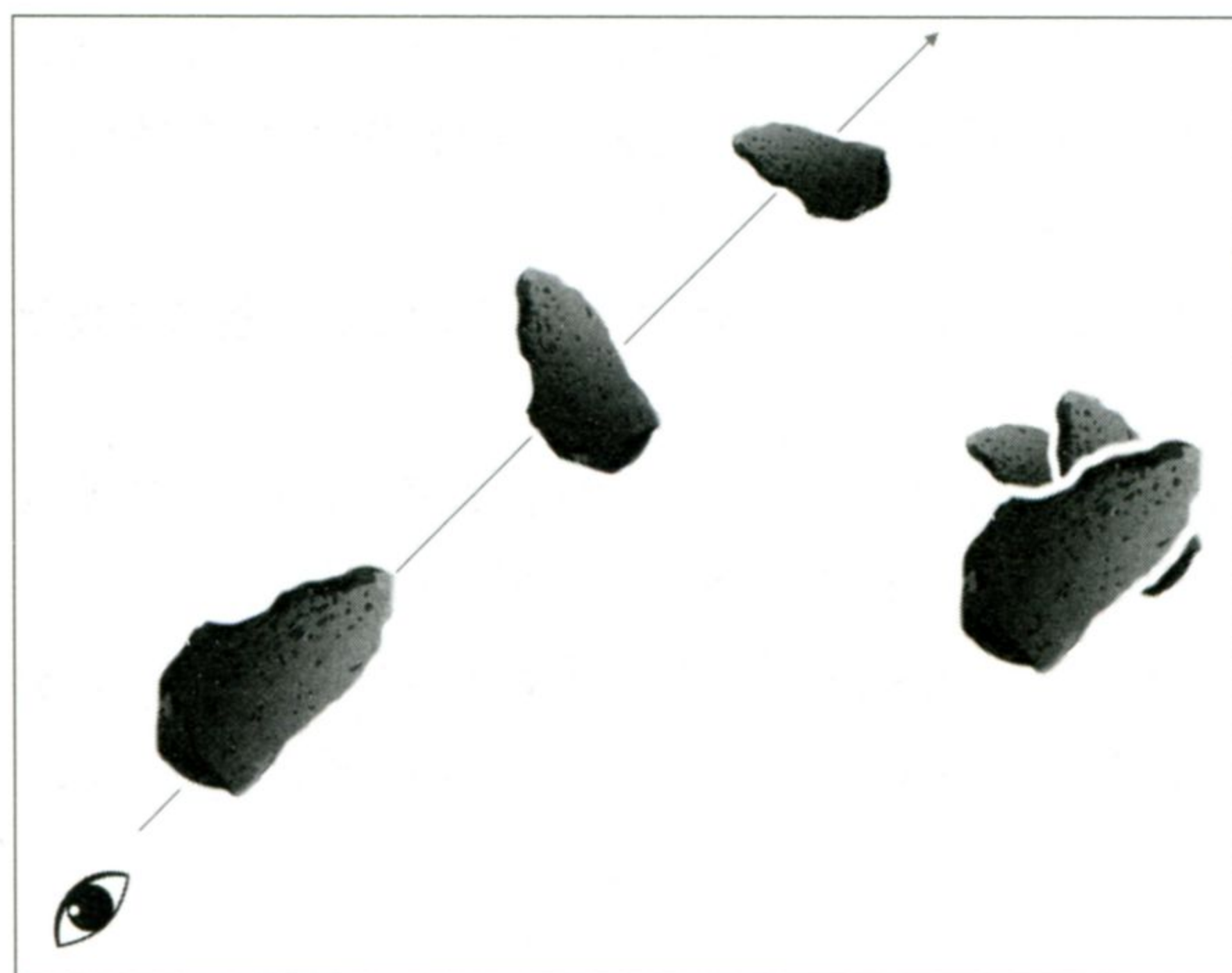
▶17 repaint()

まず背景をクリアします。18の行は説明が必要でしょう。

```
stars.sort(function (a, b) {
    return b.z - a.z;
});
```

この処理は隕石を遠い順番に並べています。なぜこのような処理が必要か考えてみましょう。ところで、ほぼ

同じ大きさの隕石が遠方から近づいてくる場合、以下のふたつのどちらが自然に見えると思いますか?

隕石の遠近感

静止画だとわかりづらいかもしれませんが、実際に動いている様子をみると右側の方が自然に感じられます。これは「近くにあるものが前面に描画されて、奥にあるものを隠している」という現実世界の状況と一致するからです。遠くの小さな隕石が、近くの大きな隕石の前に描画されるようなことは現実世界では起こりえません。その様子を以下に示します。

starsには200個の隕石が格納されていますが並び順はランダムです。奥のものから描画していけば、自然な感じに描画することができます。そこで、Z軸の値順に並べ替えていたのです。

19は隕石を描画する処理です。

```
// 隕石の描画
        for (var i = 0 ; i < 200 ; i++) {
            var star = stars[i];
            var z = star.z;
            var x = ((star.x - ship.x) << 9) / z + 400;
            var y = ((star.y - ship.y) << 9) / z + 400;      ←Ⓐ
            var size = (50 << 9) / z;
            ctx.save();
            ctx.translate(x, y);
            ctx.globalAlpha = 1- (z / 4096);      ←Ⓑ
            ctx.rotate(star.r * Math.PI / 180);
            ctx.drawImage(rockImg, -size / 2, -size / 2, size, size);
            ctx.restore();
        }
```

x、y、zの座標値の計算に若干の工夫があります。

xから見ていきましょう。自機から眺めた様子（自機を視点の中心として）を描画したいので、まず、「(star.x - ship.x)」で隕石と自機との差分を求めますⒶ。ただ、モデルとする空間は-1600から1600までの範囲という狭い範囲なので、そのままピクセルとして描画しても広がりが生まれません。

そこで、「((star.x - ship.x) << 9)」として拡大（512倍）しています。この値はいろいろためして適当に設定したものです。

ここで、「<< 9」がなぜ512倍なのかは説明が必要でしょう。この「<<」はシフト演算子といって、各ビットを指定された分だけ左方向にシフトするものです。今回の場合は9ビット左方向に移動しています。

2進数は桁上がりすると2倍になります。一番右のビットが1だったときに、左に移動するたびに、2、4、8、16…と増えることからもわかると思います。たとえば、6は二進数で110ですが、左に9個移動すると以下のように512倍されて3072となります。

各ビットを左に9個移動

```
0000 0000 0110 = 6

1100 0000 0000 = 6×512 = 3072
```

では、なぜ「 ((star.x - ship.x) * 512)」と書かなかったのでしょうか？　実はそのように書いてもまったく問題ありません。ただ、一般的にシフト演算は極めて高速に実行されます。「ちょうど500倍でないと困るんだ、512ではダメなんだ!」といった特に強いこだわりがなければ、シフト演算を使う選択肢もあるということを紹介したかったので上記のような実装にしました。

次に、512倍した値をzで割っていますが、これは遠くの隕石ほど画面中央からの差分を小さくするための処理です。+400は、800 x 800という描画領域の中心を原点とするためのものです。y軸方向の処理もx軸方向とまったく同じです。

隕石の回転描画はcanvasの章で時計を描画したときと同じ処理です。save()で現在のコンテキストを保存し、translate(x, y)で(x, y)を原点にし、rotate()で座標系を回転しています。

Ⓑは遠くの隕石ほど暗く描画するための処理です。

```
ctx.globalAlpha = 1- (z / 4096);
```

globalAlphaはこれからcanvasに描画する内容の透明度（α値）を設定するプロパティです。隕石が近くにある場合zは小さな値になるため、globalAlphaは1に近づきます。すなわち、オリジナルの画像に近い状態で描画されます。

一方、遠くにある場合、globalAlphaは0に近づき、ほとんど透明（背景の黒に溶け込む感じ）に描画されます。あとは、照準器の画像とスコア、ゲームオーバー時のテキストを描画しています。

20の行は少しトリッキーなので説明をしておきます。

```
ctx.fillText(('0000000' + score).slice(-7), 670, 60);
```

これはスコアの左を0で埋めて描画するための処理です。仮にスコアが123という数値だったとします。文字列と数値の足し算を行うと、数値は文字列に変換され、文字列の連結が行われます。

すなわち「('0000000' + score)」は「'0000000123'」という文字列になります。String.prototype.slice()は文字列の一部を切り出して返します。マイナスの数を指定した場合、文字列の末尾から切り取って返されます。つまり、左を0で埋めた文字列が取得できるのです。ゲームではこのような表示がよく行われるので真似をしてみました。

以上が疑似3Dゲームの説明です。短いコードのわりにはそれっぽい効果が演出できていると思います。自分が学生のころスペースハリアーというゲームが人気を集めていました。今思うと疑似3Dゲームに触れた最初の機会だったのかもしれません。

演習 **各パラメータを変更して確認してみよう**

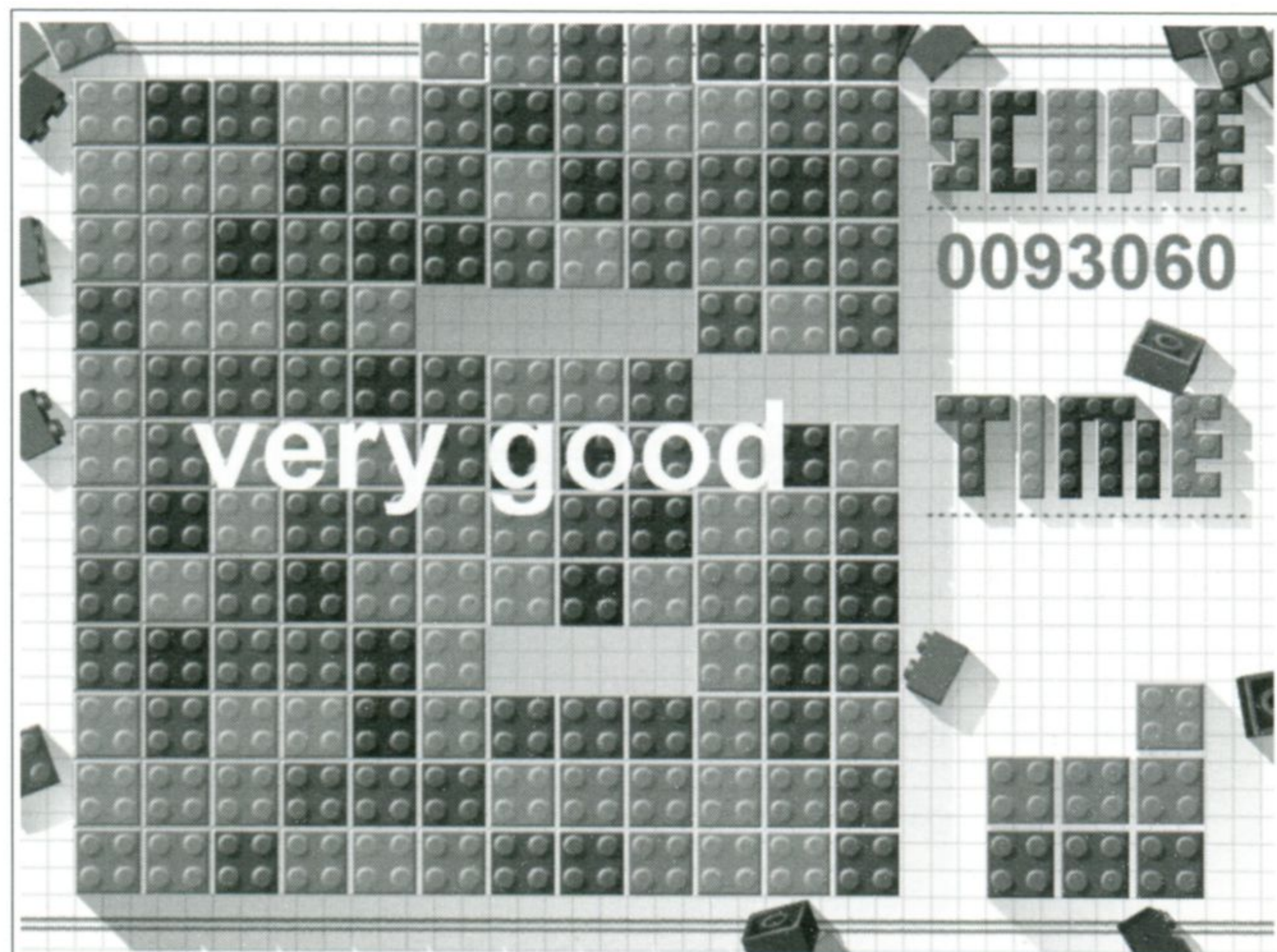
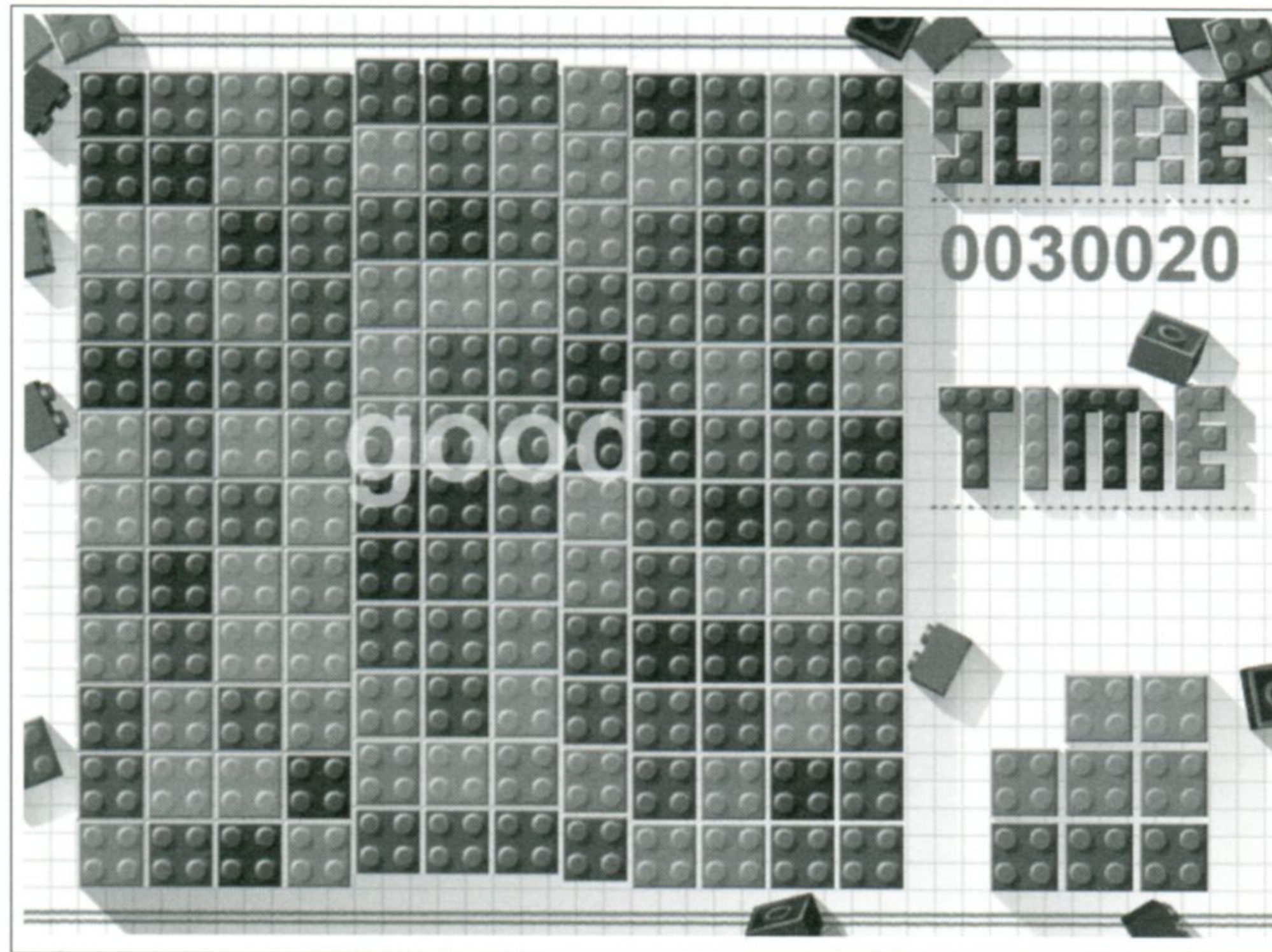
ゲームを入力した後で、512倍（<<9）やspeedといったパラメータをいろいろ変更して、それがどのように反映されるか確認してください。

5-7 Funky Blocks

いわゆる"落ちモノ系"ゲームの一例として作ってみました。縦横方向に同じ色が3つ以上揃うと消えて、上からブロックが落ちてきます。連鎖するほど高得点となります。

このゲームで学ぶこと

- 本格的なゲームを作る
- 効果音の再生方法を知る

Let's write codes !

```
<!DOCTYPE html>
<html>
<head>
    <title>FunkyBlocks</title>
    <META charset="UTF-8">
    <style>
        #canvas {
            width: 800px;
            height: 600px;
            touch-action: none;
        }
        #START {
            position: absolute;
            left: 200px;
            top: 200px;
        }
    </style>
    <script>
        "use strict";

        var ctx, tiles = [], moves = [], mIndex = 0, mCount = 0, times = [];
        var timer = NaN, startTime = NaN, elapsed = 0, score = 0, bgimage, sound;
        var mouseX = null, mouseY = null, mouseUpX = null, mouseUpY = null;
        var message = ["", "good", "very good", "super", "wondeful!", "great!!",
            "amazing", "brilliant!", "excellent!!"];

        function rand(v) {
            return Math.floor(Math.random() * v);
        }

        function iterate(f) {          ←1
            for (var x = 0 ; x < 12 ; x++) {
                for (var y = 0 ; y < 12 ; y++) {
                    f(x, y, tiles[x][y]);
                }
            }
        }

        /**
         * タイルオブジェクト
         */
        function Tile(x, y) {          ←2
            this.x = x;
```

```
        this.y = y;
        this.px = x;
        this.py = y;
        this.count = 0;
        this.getX = function () {
            return this.x + (this.px - this.x) * (this.count) / 20;
        }
        this.getY = function () {
            return this.y + (this.py - this.y) * (this.count) / 20;
        }
        this.move = function (px, py, color) {
            this.px = px;
            this.py = py;
            this.color = color;
            this.count = 20;
            this.moving = true;
            moves.push(this);
        }
        this.update = function () {
            if (--this.count <= 0) {
                this.moving = false;
            }
        }
    }

    function init() {          ←3
        // タイルオブジェクトの生成
        for (var x = 0 ; x < 12 ; x++) {
            tiles[x] = [];
            for (var y = 0 ; y < 12 ; y++) {
                tiles[x][y] = new Tile(x, y);
            }
        }

        // 3つ連続しないよう初期色の配置
        iterate(function (x, y, t) {
            while (true) {
                var r = rand(5);
                if (setColor(x, y, r)) {
                    t.color = r;
                    break;
                }
            }
        });
```

```
    // 残り時間初期化
    for (var i = 0 ; i < 15 ; i++) {
        var t = document.createElement("img");
        t.src = "time" + i + ".png";
        times.push(t)
    }

    // Canvas初期化
    bgimage = document.getElementById("bgimage");
    var canvas = document.getElementById("canvas");
    ctx = canvas.getContext("2d");
    ctx.textAlign = "center";

    sound = document.getElementById("sound");
    repaint();
}

function go() {        ←4
    var canvas = document.getElementById("canvas");
    canvas.onmousedown = mymousedown;
    canvas.onmouseup = mymouseup;
    canvas.addEventListener('touchstart', mymousedown);
    canvas.addEventListener('touchmove', mymousemove);
    canvas.addEventListener('touchend', mymouseup);

    startTime = new Date();
    timer = setInterval(tick, 25);

    document.body.addEventListener('touchmove', function (event) {
        event.preventDefault();
    }, false);
    document.getElementById("START").style.display = "none";
    document.getElementById("bgm").play();
}

/**
 * メインループ
 */
function tick() {        ←5
    // メッセージフェードアウト効果
    mCount = Math.max(0, mCount - 1);
    if (mCount == 0) {
        mIndex = 0;
    }
```

```
        if (moves.length > 0) {
            for (var i = 0 ; i < moves.length ; i++) {  // タイル移動
                moves[i].update();
            }
            moves = moves.filter(function (t) { return t.count != 0 });
            if (moves.length == 0) {    // 移動完了
                var s = removeTile();   // タイル消去
                if (s > 0) {
                    // 初回 or 連鎖
                    mIndex = Math.min(message.length - 1, mIndex + 1);
                    mCount = 50;
                    score += s * 10 + mIndex * s * 100;
                    sound.pause();
                    sound.currentTime = 0;
                    sound.play();
                }
                fall();
            }
        }

        elapsed = ((new Date()).getTime() - startTime) / 1000;
        if (elapsed > 69) {
            clearInterval(timer);
            timer = NaN;
        }
        repaint();
    }

    function setColor(x, y, c) {          ←6
        var flag = true;
        if (1 < x) {    // 左
            var c0 = tiles[x - 2][y].color;
            var c1 = tiles[x - 1][y].color;
            flag &= !(c0 == c1 && c1 == c);
        }
        if (x < 8) {    // 右
            var c0 = tiles[x + 2][y].color;
            var c1 = tiles[x + 1][y].color;
            flag &= !(c0 == c1 && c1 == c);
        }
        if (1 < y) {    // 上
            var c0 = tiles[x][y - 2].color;
            var c1 = tiles[x][y - 1].color;
            flag &= !(c0 == c1 && c1 == c);
        }
```

```
    if (y < 8) {    // 下
        var c0 = tiles[x][y + 2].color;
        var c1 = tiles[x][y + 1].color;
        flag &= !(c0 == c1 && c1 == c);
    }
    return flag;
}

function mymousedown(e) {          ←7
    mouseX = !isNaN(e.offsetX) ? e.offsetX : e.touches[0].clientX;
    mouseY = !isNaN(e.offsetY) ? e.offsetY : e.touches[0].clientY
}

function mymousemove(e) {          ←8
    mouseUpX = !isNaN(e.offsetX) ? e.offsetX : e.touches[0].clientX;
    mouseUpY = !isNaN(e.offsetY) ? e.offsetY : e.touches[0].clientY
}

function mymouseup(e) {            ←9
    var sx = Math.floor((mouseX - 34) / 44);
    var sy = Math.floor((mouseY - 36) / 44);
    var nx = sx, ny = sy;
    var mx = !isNaN(e.offsetX) ? e.offsetX : mouseUpX;
    var my = !isNaN(e.offsetY) ? e.offsetY : mouseUpY;
    if (Math.abs(mx - mouseX) > Math.abs(my - mouseY)) {
        nx += (mx - mouseX > 0) ? 1 : -1;
    } else {
        ny += (my - mouseY > 0) ? 1 : -1;
    }

    if (nx > 11 || ny > 11 || nx < 0 || ny < 0 ||
        tiles[sx][sy].moving || tiles[nx][ny].moving) {
        return
    }

    var c = tiles[sx][sy].color;
    tiles[sx][sy].move(nx, ny, tiles[nx][ny].color);
    tiles[nx][ny].move(sx, sy, c);
    repaint();
}

function removeTile() {            ←10
    // 縦横3つ以上連続するタイルにremoveフラグをセット

    for (var y = 0 ; y < 12 ; y++) {    // 横方向
```

```
        var c0 = tiles[0][y].color;
        var count = 1;
        for (var x = 1 ; x < 12 ; x++) {
            var c1 = tiles[x][y].color;
            if (c0 != c1) {
                c0 = c1;
                count = 1;
            } else {
                if (++count >= 3) {
                    tiles[x - 2][y].remove = true;
                    tiles[x - 1][y].remove = true;
                    tiles[x - 0][y].remove = true;
                }
            }
        }
    }

    for (var x = 0 ; x < 12 ; x++) {    // 縦方向
        var c0 = tiles[x][0].color;
        var count = 1;
        for (var y = 1 ; y < 12 ; y++) {
            var c1 = tiles[x][y].color;
            if (c0 != c1) {
                c0 = c1;
                count = 1;
            } else {
                if (++count >= 3) {
                    tiles[x][y - 2].remove = true;
                    tiles[x][y - 1].remove = true;
                    tiles[x][y - 0].remove = true;
                }
            }
        }
    }
    var score = 0;
    iterate(function (x, y, t) { if (t.remove) { score++ } });
    return score;
}

function fall() {   // 落下処理        ←11
    for (var x = 0 ; x < 12 ; x++) {
        for (var y = 11, sp = 11; y >= 0 ; y--, sp--) {
            while (sp >= 0) {
                if (tiles[x][sp].remove) {
                    sp--;
```

```
                } else {
                    break;
                }
            }
            if (y != sp) {
                var c = (sp >= 0) ? tiles[x][sp].color : rand(5);
                tiles[x][y].move(x, sp, c);
            }
        }
    }
    iterate(function (x, y, t) { t.remove = false; })
}

function repaint() {           ←12
    ctx.drawImage(bgimage, 0, 0);

    // タイル
    var images = [block0, block1, block2, block3, block4];
    iterate(function (x, y, t) {
        if (!t.remove) {
            ctx.drawImage(images[t.color], t.getX() * 44 + 34, t.getY() * 44 + 36,
42, 42)
        }
    });

    // メッセージ
    ctx.font = "bold 80px sans-serif";
    ctx.fillStyle = "rgba(255, 255, 255, " + (mCount / 50) + ")";
    ctx.fillText(message[mIndex], 300, 300);
    ctx.fillStyle = "white";

    if (isNaN(timer)) {
        ctx.fillText("FINISH", 350, 300);
    }

    // スコア
    ctx.fillStyle = "rgba(220, 133, 30, 50)";
    ctx.font = "bold 50px sans-serif"
    ctx.fillText(('0000000' + score).slice(-7), 680, 170);

    // 残り時間
    var index = Math.min(15, Math.floor(elapsed / (69 / 15)));
    ctx.drawImage(times[index], 615, 327);
}
</script>
```

```
</head>
<body onload="init()">
    <!-- Thanks to http://takao-suenobu.com/  & http://dova-s.jp/ -->
    <audio src="sound.mp3" id="sound"></audio>
    <audio src="letsgo.mp3" id="bgm"></audio>
    <canvas id="canvas" width="800" height="600"></canvas>
    <img id="START" src="start.png" onclick="go()"><br/>
    <img id="bgimage" src="back.png" style="display:none" />
    <img id="block0" src="block0.png" style="display:none" />
    <img id="block1" src="block1.png" style="display:none" />
    <img id="block2" src="block2.png" style="display:none" />
    <img id="block3" src="block3.png" style="display:none" />
    <img id="block4" src="block4.png" style="display:none" />
</body>
</html>
```

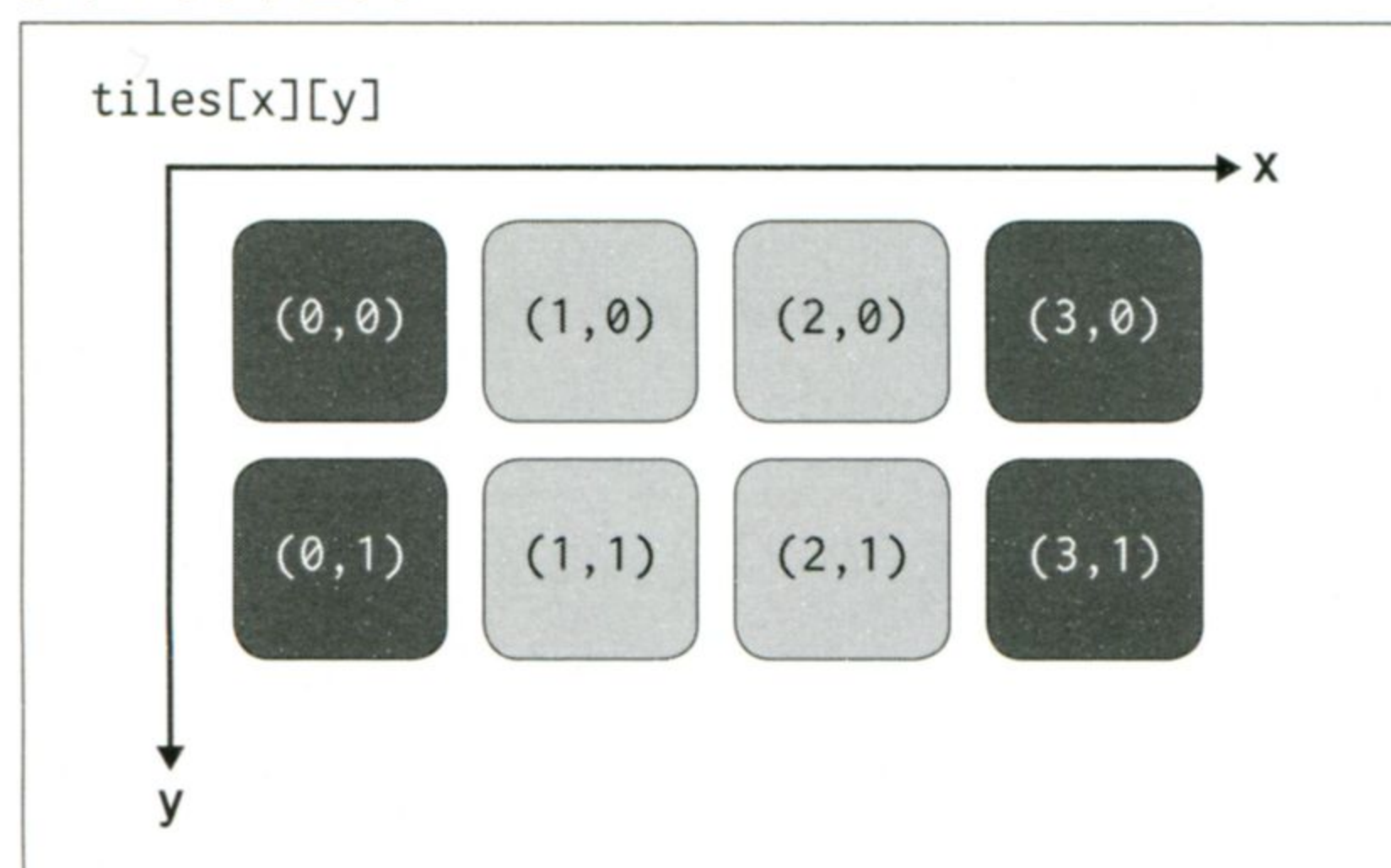
5-7-1 ｜ソースコード解説

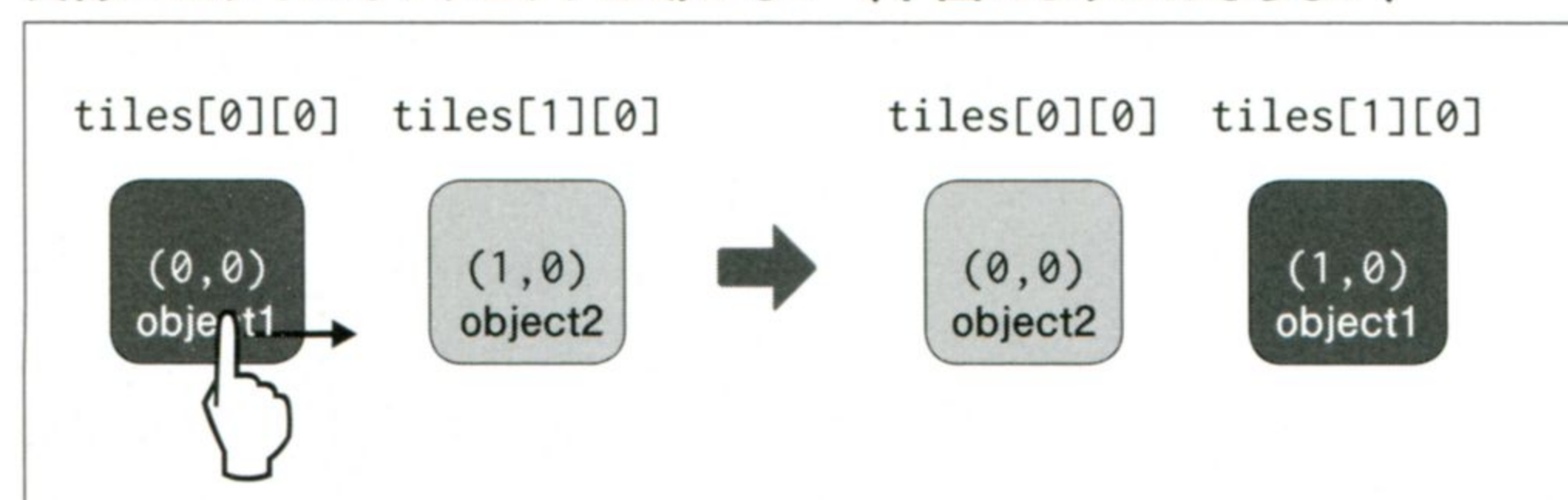
使用している広域変数は以下の通りです。

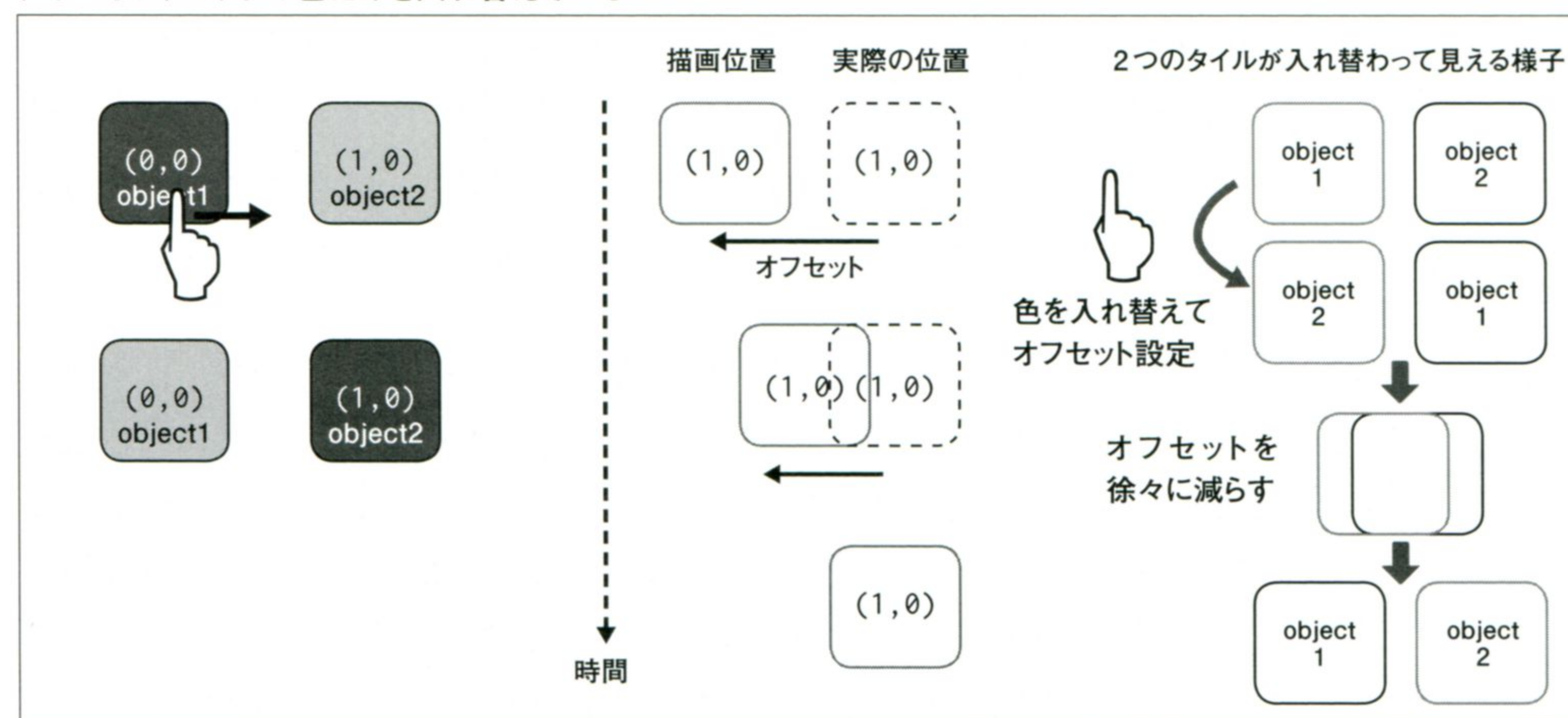
使用している広域変数

変数	説明
ctx	グラフィックコンテキスト
tiles	タイルオブジェクトを格納する二次元配列
moves	移動中のタイルを保持する配列
mIndex	メッセージへのインデックス（=何連鎖中かを保持）
mCount	メッセージフェードアウト効果を演出するためのカウンタ
times	残り時間画像を格納する配列
timer	タイマー
startTime	ゲーム開始時刻
elapsed	経過時間
score	スコア
bgimage	背景画像
sound	ブロックが消えたときの効果音
mouseX	マウス押下時のＸ座標
mouseY	マウス押下時のＹ座標
mouseUpX	マウスリリース時のＸ座標
mouseUpY	マウスリリース時のＹ座標
message	メッセージの配列

このゲームで中心的な役割をするのはタイルオブジェクトです。タイルは以下のように2次元配列として実装されています。

タイルオブジェクト

ゲームの進行に伴ってタイルオブジェクトも移動するように見えますが、実際にはタイルオブジェクトは動きません。つまり、仮に(0,0)のタイルを右に移動させても以下のようにはなりません。

実際にはタイルオブジェクトは動かない（下図のようにはならない）

代わりに、タイルオブジェクトの色だけを入れ替えています。

タイルオブジェクトの色だけを入れ替えている

(0,0)のタイルを右に移動する操作を行ったとします。このときは(0,0)と(1,0)の色を入れ替えます。しかし、単に色を入れ替えただけでは、スムーズに動く視覚効果が得られません。そこで、上中央図にあるように描画位置のオフセットをカウンタで調整し、入れ替える2つのタイルにこの効果を適用しています。こうすることで上右図にあるように2つのタイルのスムーズな入れ替えを演出しています。

タイルはTileオブジェクトとして実装しています**2**。

```
function Tile(x, y) {
    this.x = x;
    this.y = y;
    this.px = x;
    this.py = y;
    this.count = 0;
    this.getX = function () {
        return this.x + (this.px - this.x) * (this.count) / 20;
    }
    this.getY = function () {
        return this.y + (this.py - this.y) * (this.count) / 20;
    }
    this.move = function (px, py, color) {
        this.px = px;
        this.py = py;
        this.color = color;
        this.count = 20;
        this.moving = true;
        moves.push(this);
    }
    this.update = function () {
        if (--this.count <= 0) {
            this.moving = false;
        }
    }
}
```

オブジェクは現在の座標(x, y)、移動先の座標(px, py)、移動までのカウンタcountをプロパティとして持ちます。getXとgetYは描画座標を返すメソッドです。countの値によってオフセットを調整しています。countに0や20といった値を代入してどんな値が返るか考えてみると理解しやすいかもしれません。「move(px, py, color)」はタイルを視覚的に移動させるメソッドです。移動先の座標と色を指定します。countを20で初期化し、movingフラグをtrueにし、移動タイル格納配列movesに自分を挿入します。updateはカウンタをデクリメント（=1減らす）し、0になったときにフラグをfalseに戻します。個々の処理はそれほど複雑ではありません。

では主な関数を見ていきましょう。

▶1 iterate(f)

引数に関数オブジェクトを取り、すべてのタイルについてその関数を実行します。このゲームでは、3つ並んでいないか検査する、個々のタイルが削除対象か否か調べる、個々のタイルを描画するといった処理を行う必要がありました。いずれも2重ループを使ってすべてのタイルを処理する必要がありますが、個々の処理内容は異なります。そこで、共通部分である2重ループのみをiterate(f)で括りだし、個別の処理を関数オブジェクトで指定するように実装しました。これにより2重ループの数を減らすことができました。

▶3 init()

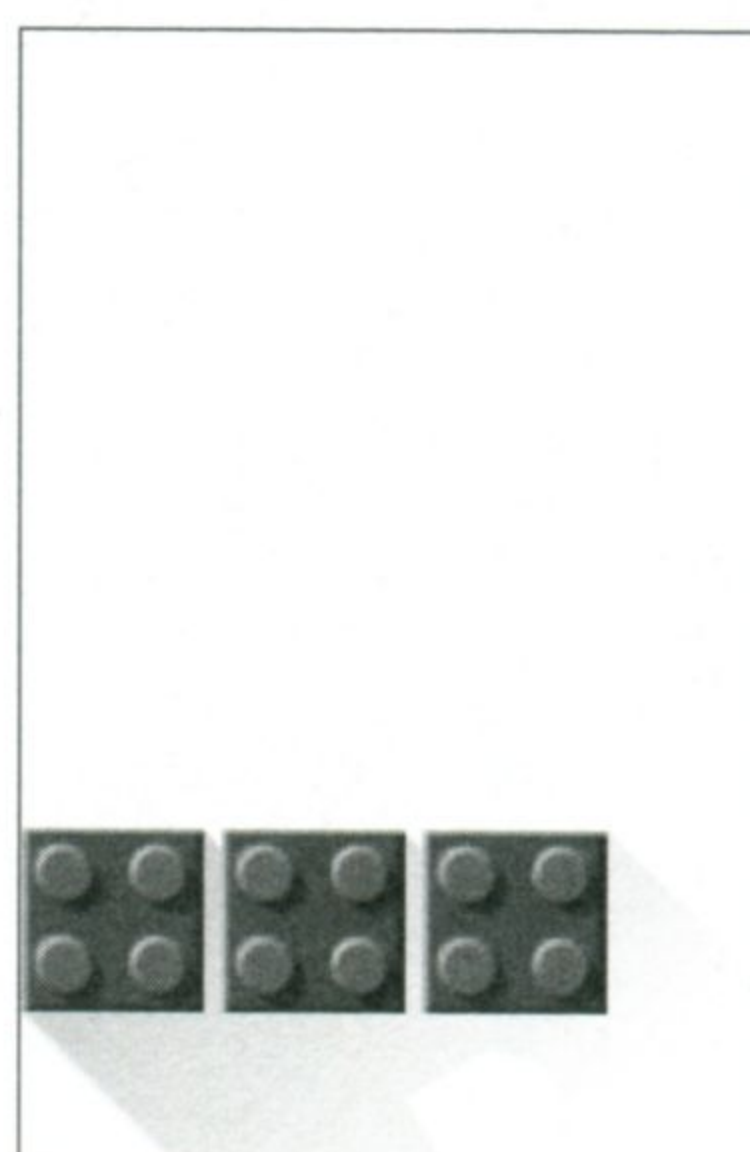
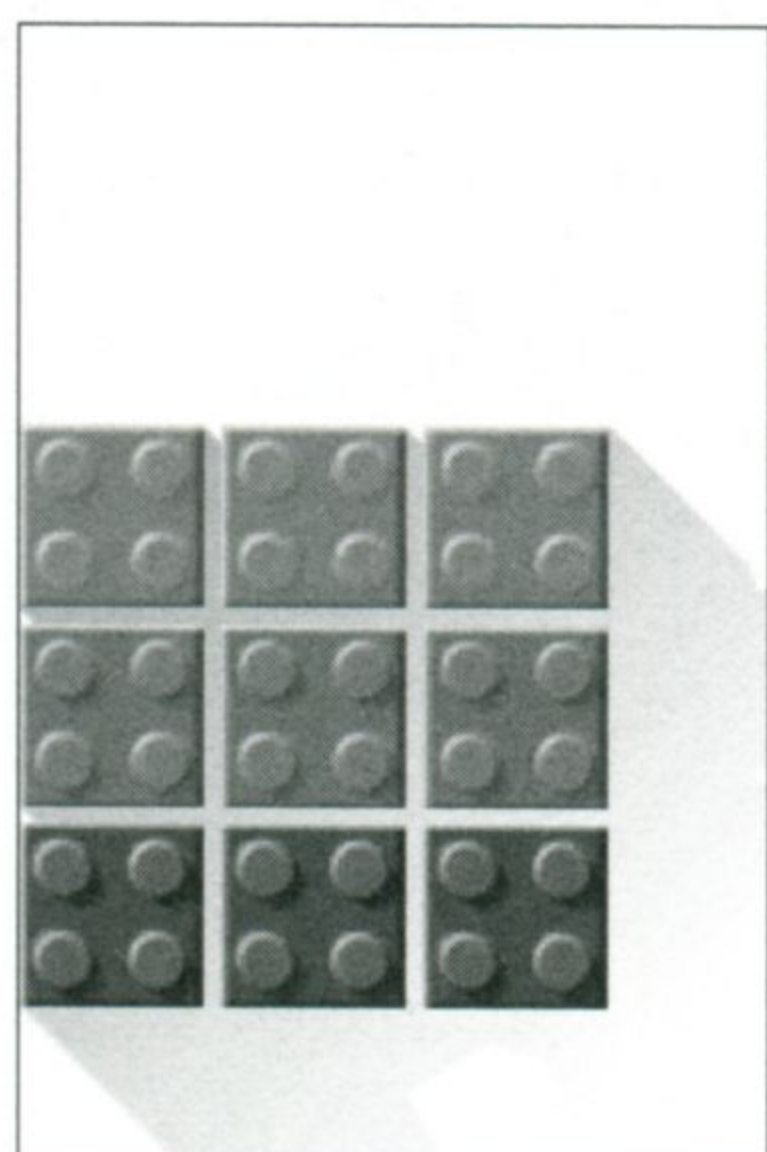
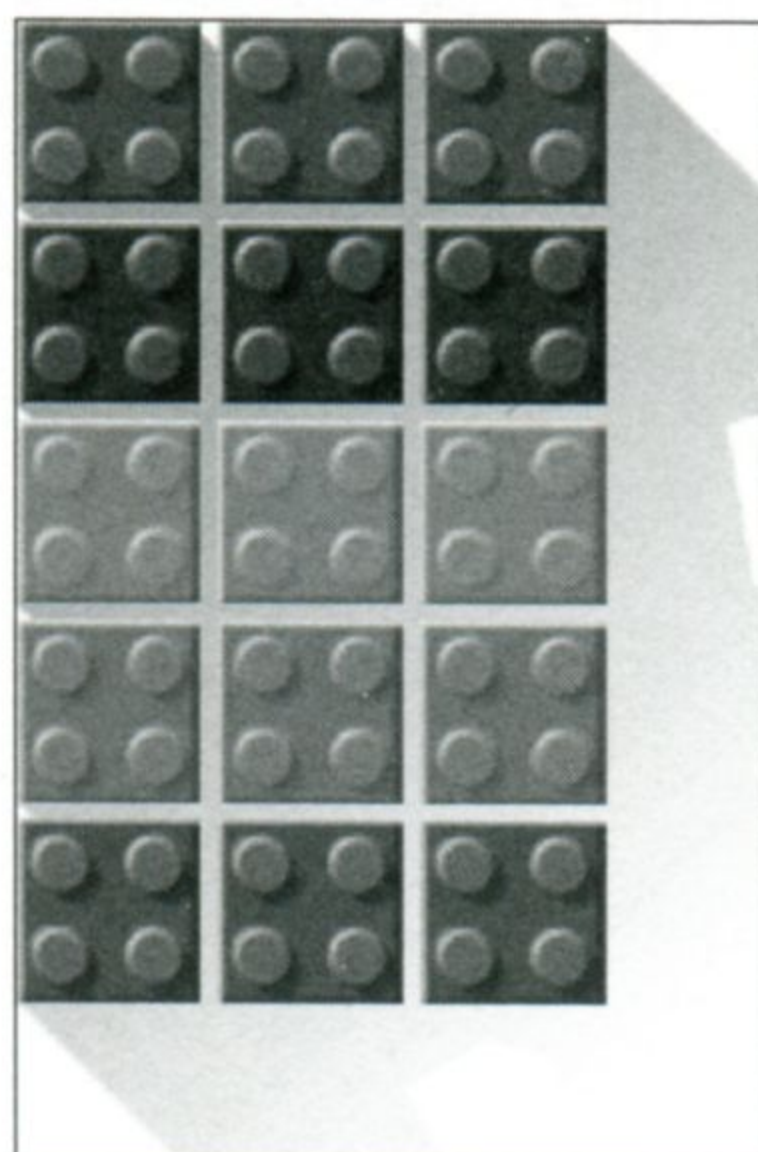
まず2重ループでタイルオブジェクトの二元配列を作成します。その後で、各タイルに色を設定しますが、最初の段階では上下左右に3つ以上同じ色がならばないようにしなくてはなりません。その処理を行っているのが以下の部分です。

```
iterate(function (x, y, t) {
    while (true) {
        var r = rand(5);
        if (setColor(x, y, r)) {
            t.color = r;
            break;
        }
    }
});
```

関数オブジェクトを引数としてiterateを呼び出しています。これにより、すべてのタイル位置について関数が呼び出されます。関数オブジェクトの引数はx座標、y座標、タイルオブジェクトtとなります。色を5色から乱数で選び、setColorで色を設定します。設定できた場合はtrueがかえるので次のタイルの色設定に進みます。falseが帰った場合は色が連続しているので再度乱数の生成から処理を行います。この関数オブジェクトが呼び出される様子は、コードを見ているだけでは分かりにくいのでデバッガを使って動きをみることをお勧めします。

次に残り時間の画像の配列を作成し、times配列に格納します。あとはcanvasのコンテキストを設定し、repaint()で描画を行っています。

残り時間はブロックを徐々に消すことで表現しています。経過時間に応じてJavaScriptで描画するブロックの数を変えてもよかったのですが、デザイナーの方が15枚の画像を作成してくださったので、それをそのまま使用することにしました。

ブロックの画像

よく見るとブロックの数に応じて影が変化していることがわかります。このような細部へのこだわりはゲームの完成度を高めるうえでとても大切だと思いました。

▶ 4 go()

ゲームを実際に開始する関数です。STARTボタンの押下で呼び出されます。まず、canvasにマウスやタッチのイベントハンドラを登録します。そして、メインループtickを開始し、STARTボタンを非表示にして、BGMの再生を開始しています。

▶ 5 tick()

ゲームの心臓部分ともいえるメインループです。25msecごとに呼び出されます。まず、

```
mCount = Math.max(0, mCount - 1);
if (mCount == 0) {
     mIndex = 0;
}
```

でメッセージフェードアウト効果を演出しています。mCountは連鎖時に表示されるメッセージの色の濃さです。Math.max(0, mCount - 1)はマイナスにならないように単にmCountを1減らしています。mCountが0になったときはメッセージが消えたときなので、mIndexを0にして連鎖をクリアしています。次のif文の条件式「(moves.length > 0)」は移動中のタイルの有無を判定しています。移動中のタイルがある場合には、「moves[i].update()」ですべてのタイルの状態を更新します。

```
moves = moves.filter(function (t) { return t.count != 0 });
```

では、移動中のタイルのみを抽出しています。Arrayオブジェクトのfilterメソッドは、引数の関数がtrueを返した要素を選択（フィルタリング）して、配列として戻します。つまり、タイルのcountが0でないもの、すなわち、まだ移動中のタイルが配列としてmovesに戻されます。

「(moves.length == 0)」がtrueの場合は、すべてのタイルが移動を完了したので、removeTile()を呼び出して3つ以上連続したタイルの消去処理を行います。この関数は消去したタイルの枚数を返します。その数が0より大きい場合は連鎖となるので、音を鳴らしメッセージのインデックスmIndexや、メッセージのフェードアウトカウンタmCountを初期化し、スコアを加算します。

あとは、ゲーム開始からの経過時間を以下の式でもとめ、69秒（BGMの再生時間）を超えたときにゲーム終了としています。

```
elapsed = ((new Date()).getTime() - startTime) / 1000;
```

▶6 setColor(x, y, c)

初期化時に色を設定する関数です。上下左右に3つ以上同じ色が隣り合わない場合にtrueを、そうでない場合はfalseを返します。左右上下の4方向を調べていますが、端に近い場合は連続するタイルがないので、4つのif文を使ってその条件を調べています。

「flag &= !(c0 == c1 && c1 == c)」は少し説明が必要でしょう。引数の色がc、隣の色がc1、さらに隣の色がc0です。この3つの色が同じとき「(c0 == c1 && c1 == c)」がtrueになります。その前にある「!」は否定演算子で、trueとfalseを逆転させます。

つまり、この行は「flag = flag & (3つの色がすべて同じではない)」という処理を行っているのと同じです。ここで&はAND演算で、両方がtrueの時のみ結果がtrueとなります。よって、3つの色が同じだと、「flag = flag & false」となり、flagもfalseとなります。この比較を左右上下4方向について行い、すべての方向で3つ色が揃っていないときのみtrueが返るようにしています。

▶7 mymousedown(e)

マウス押下時、タッチ操作時の座標をmouseXとmouseYに格納します。

▶8 mymousemove(e)

タイルを移動する場合、マウス押下時とリリース時の座標を比較して、どの方向に操作されたか調べています。ほとんどの場合、リリース時の座標はmymouseup(e)で取得できるのですが、特定のブラウザでタッチ操作が行われた場合に座標値が取得できないことがありました。そのような状況に対処するため、最後に移動した際の座標をmymousemove(e)で取得し、「(mouseUpX, mouseUpY)」に保存しています。

▶ 9 mymouseup(e)

マウスが離されたときのコールバックです。タイル領域の左上座標は(34, 36)、タイル1つ分の幅と高さが44です。以下の式でマウス押下時にどのタイルが選択されたかを調べます。

```
var sx = Math.floor((mouseX - 34) / 44);
var sy = Math.floor((mouseY - 36) / 44);
```

次に、タッチやマウスの操作方向を調べます。押下時の座標は(mouseX, mouseY)です。リリース時の座標は(mx, my)です。これらの座標を比較し、x軸方向、y軸方向のどちらに大きく移動しているか調べます。

```
if (Math.abs(mx - mouseX) > Math.abs(my - mouseY)) {
```

がtrueのときは、y軸方向よりもx軸方向への移動量が多いことになります。場合、移動方向に応じてnxを1か-1に設定します。y軸方向でも同様の処理を行います。

移動先の座標が範囲外の場合、もしくは、タイルが移動中である場合はreturnで戻ります。

```
if (nx > 11 || ny > 11 || nx < 0 || ny < 0 ||
    tiles[sx][sy].moving || tiles[nx][ny].moving) {
    return
}
```

そうでない場合は、以下の行でそれぞれのタイルの場所と色を入れ替える処理を開始します。

```
var c = tiles[sx][sy].color;
tiles[sx][sy].move(nx, ny, tiles[nx][ny].color);
tiles[nx][ny].move(sx, sy, c);
```

▶ 10 removeTile()

横方向、縦方向に3つ以上同じ色が並んだときに、そのタイルオブジェクトのremoveプロパティにtrueを設定します。

```
for (var y = 0 ; y < 12 ; y++) {     // 横方向
    var c0 = tiles[0][y].color;
    var count = 1;
```

で上から1行ずつ調べていきます。左端の色をc0に、連続数を示すcountを1に初期化します。

```
for (var x = 1 ; x < 12 ; x++) {
    var c1 = tiles[x][y].color;
    if (c0 != c1) {
        c0 = c1;
        count = 1;
    } else {
        if (++count >= 3) {
            tiles[x - 2][y].remove = true;
```

次に横方向に調べます。「var x = 1」として隣のタイルから調べていることに注意してください。そのタイルの色をc1に格納し、色が違う場合(c0 != c1)、c0とcountをクリアします。色が同じ場合はelseに進み、countの値が3以上になったら該当するタイルのremoveプロパティをtrueにしています。removeプロパティはもともとTileオブジェクトにはなかったものです。JavaScriptではこのように実行中にプロパティを追加･削除することが可能です。

縦方向についても同様の処理を行います。最後に以下の行でremoveプロパティがtrueに設定されたタイルの数を数えています。

```
iterate(function (x, y, t) { if (t.remove) { score++ } });
```

▶11 fall()

落下処理を行います。コードは短いのですが今回の関数の中で一番複雑な処理かもしれません。落下なのでx軸方向への移動はありません。そこで外側のループをx軸方向として、左列から右列へ順番に処理を行います。

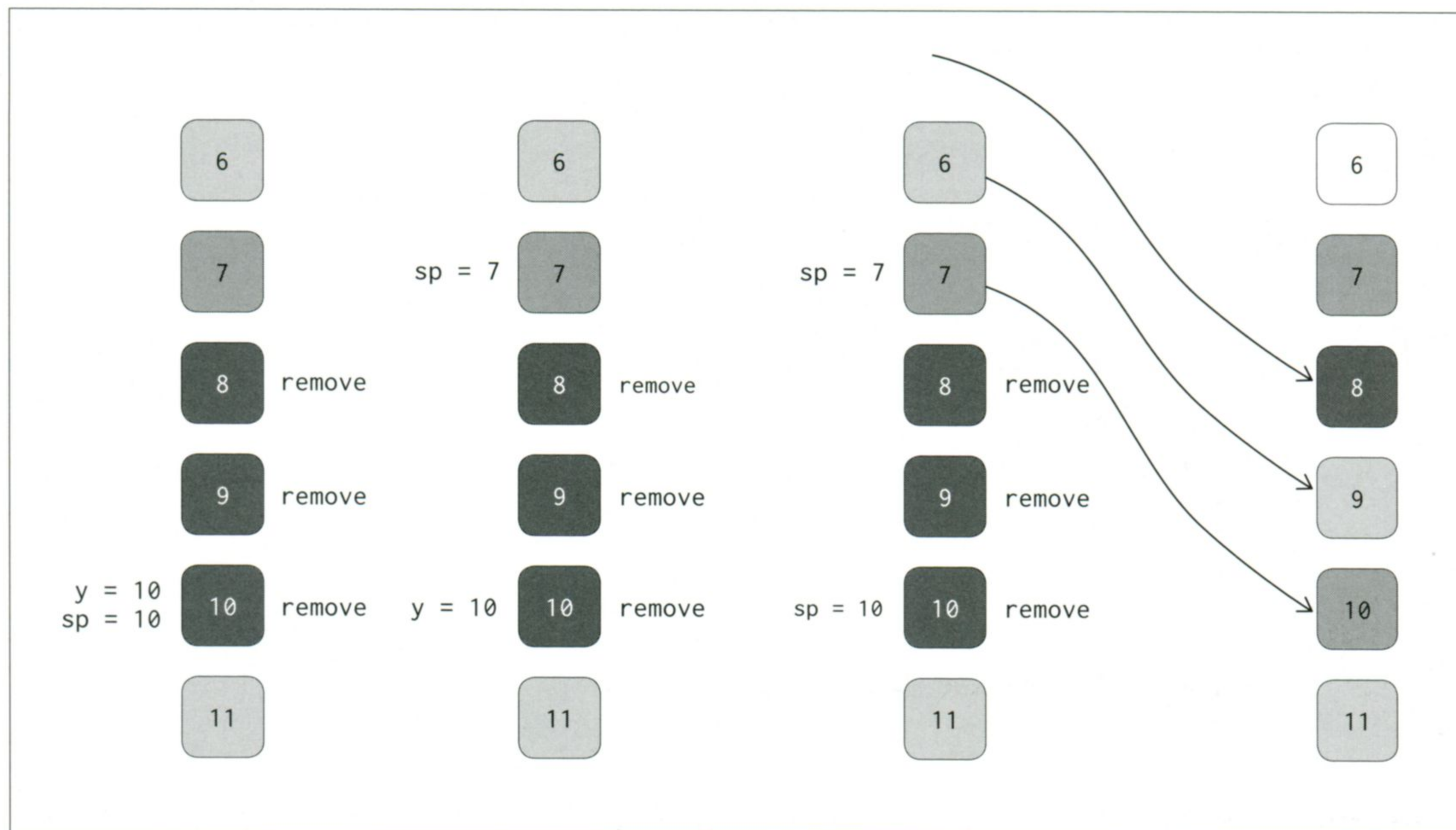
落下処理なのでy軸方向は下から上方向へ見ていきます。yは現在のタイルを示す値で11から0まで順番に減らします。一方spは、落下元のタイルを示す値です。削除されたタイルをスキップして減少させます。removeTile()で削除されたタイルにremoveプロパティを設定したことを思い出してください。この値が異なる場合にタイルが落下することになります。その様子を以下に示します。

タイル落下の様子

6
7
8 remove
9 remove
y = 10
sp = 10
10 remove
11

6
sp = 7 7
8 remove
9 remove
y = 10 10 remove
11

6
sp = 7 7
8 remove
9 remove
sp = 10 10 remove
11

6
7
8
9
10
11

yもspも11で初期化し、ループを実行する都度デクリメントします。removeのタイルがある場合、spはそれをスキップするためにさらにデクリメントします。その処理が以下の部分です。

```
while (sp >= 0) {
      if (tiles[x][sp].remove) {
          sp--;
      } else {
          break;
      }
}
```

yとspとの値が違う場合、タイルが削除されたことを意味します。よって、移動元tiles[x][sp]を、移動先tiles[x][y]にmoveします。spが0未満のときは落ちてくるタイルがないので乱数で色を決めています。

この時点で落下処理がおわったので、「iterate(function (x, y, t) { t.remove = false; })」で、removeプロパティにfalseを代入しています。

▶12 repaint()

ここまで読み進めてきた読者であれば詳しい説明はいらないはずです。背景画像を描画し、タイルを描画し、必要に応じてメッセージを描画しています。メッセージに濃度を以下の行で調整しています。

```
ctx.fillStyle = "rgba(255, 255, 255, " + (mCount / 50) + ")";
```

また、残り時間は配列に格納した画像で表現しています。

```
var index = Math.min(15, Math.floor(elapsed / (69 / 15)));
ctx.drawImage(times[index], 615, 327);
```

少し長めのコードにはなりましたが、330行程度という長さの割には見た目も良い面白いゲームに仕上がったのではないでしょうか。

演習　ゲームに工夫を加えて完成度を高めてみよう

背景画像を描画する、メッセージを工夫する、連鎖による加点を凝ったものにする、まだまだ改良の余地はたくさんあります。自分なりの工夫を加えることでより高い完成度のゲームに仕上げてください。

Chapter 6

HTML5
CSS
JavaScript
Canvas
Game
and
Physics engine

物理エンジンを使ったゲーム

本書の締めくくりとして物理エンジンを使ったゲームを紹介します。ここでは一般的に使われている物理エンジンを利用するのではなく、物理エンジンそのものから実装します。なお、物理エンジン自体の説明は、残念ながら紙面の都合上掲載できませんでした。物理エンジンがどのように作られているか、その中身にも興味がある方は以下のURLにある解説記事を参照してください。

https://thinkit.co.jp/series/4770

6-1 物理エンジンとは

物理エンジンとは、さまざまな物理法則をシミュレートし、物体の衝突や動きを計算するものです。これを利用することで実世界のようなリアルな動きを再現できます。使い方は簡単です。工夫次第でいろいろなゲームを作ることができるのでぜひ試してみてください。

6-1-1 | なぜ物理エンジンを使おうと思ったか?

「Angry Bird」や「Cut the Rope」など物理エンジンを利用したパズル系のゲームが人気です。「より多くの読者に興味を持ってもらうために物理エンジンを使ったリアルなゲームを作りたい、しかしながら、既存のライブラリを使ってもそれだけで本1冊位のボリュームになってしまう」と悩みました。

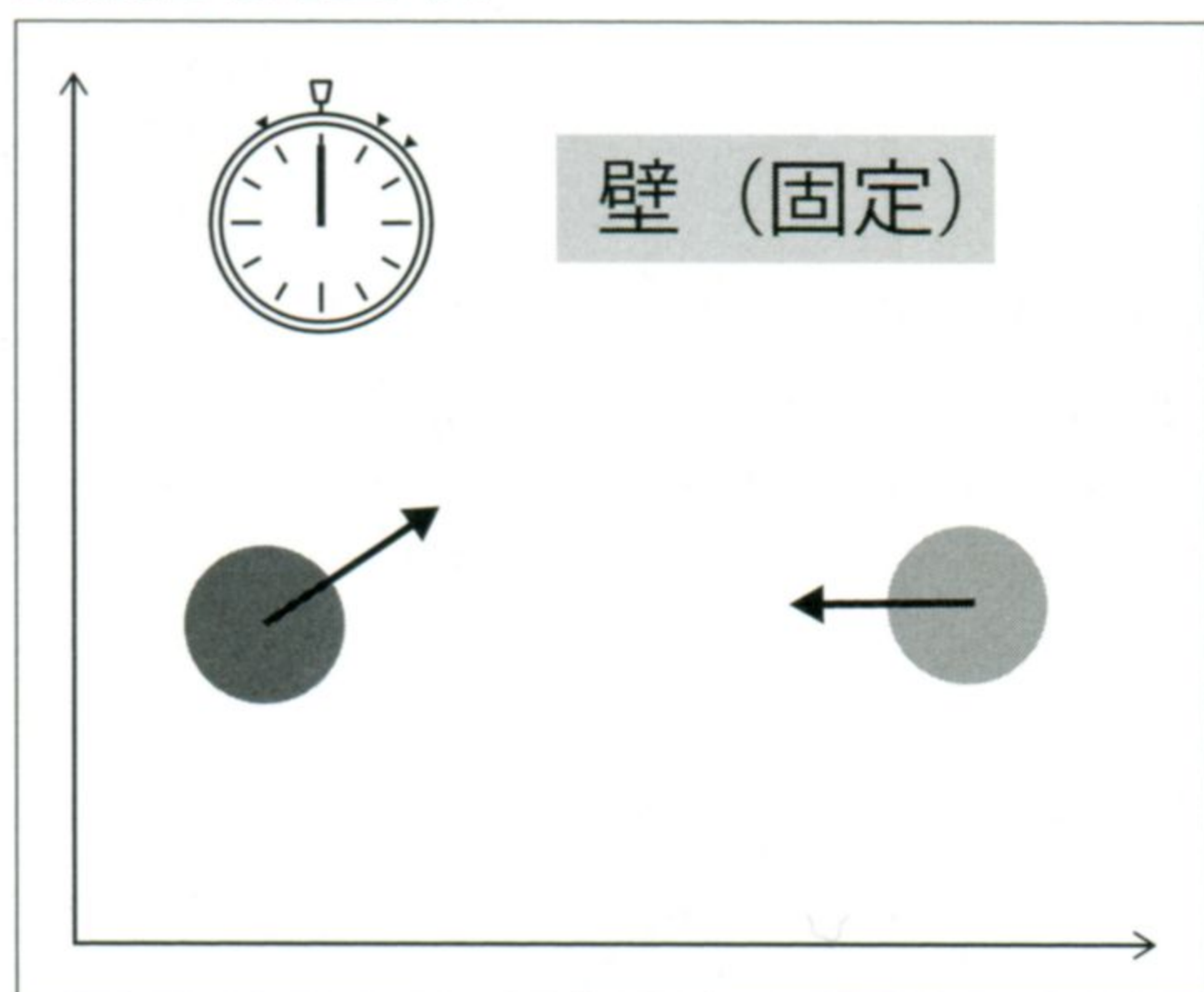
リアルな動きを再現するためには、摩擦、衝突、慣性、重力、運動量保存、重心、角速度といったさまざまな物理法則に基づいた計算を行う必要がありますが、その実装は容易ではありません。そこで、多くのゲームは既存の物理エンジンライブラリを利用しています。代表的なライブラリにBox2D、PhysicsJS、ThreeJS、Unityなどがあります。ライブラリを使うと複雑な物理計算を自分で行う必要がなくなりますが、それでもライブラリの習得には時間と労力がかかります。

そこで、本書では極限までシンプルな物理エンジンを自作することにしました。角速度や質量は考慮しない、矩形・円・線しかサポートしないなど、物理エンジンと名乗るには僭越なほどシンプルなものです。行数はたった250程度です。しかしながら、シンプルなだけに使いやすく、修正や拡張も簡単です。まさに「百聞は一見にしかず」です。まずはサンプルを実行してみて、どのような動きをするのか確かめてください。“どんなふうに作られているのだろう”と興味をもっていただけたなら、今年の休暇をほとんど費やした自分の努力が報われたことになります。

6-1-2 | 物理エンジンの仕組み

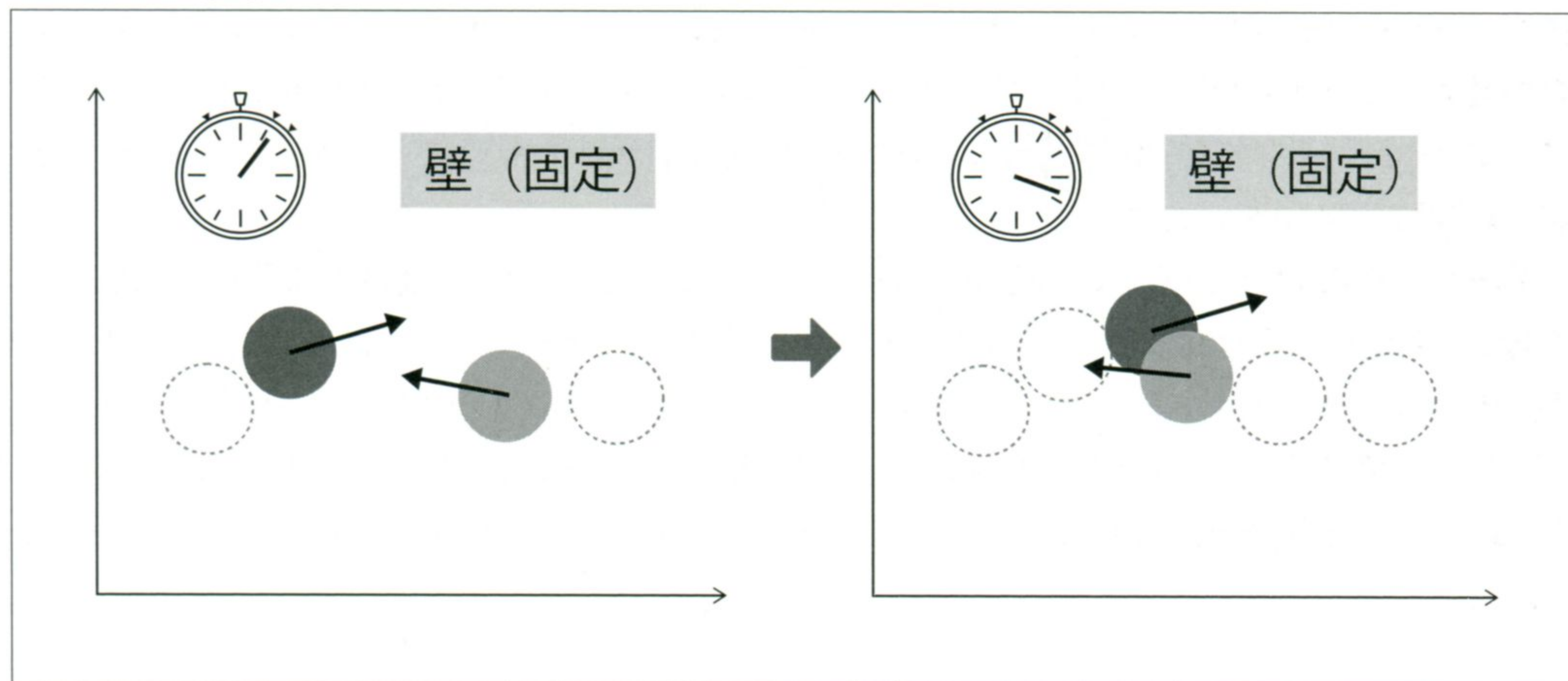
物理エンジンを利用する前に、その概要について説明しておきます。一般的な物理エンジンでは、最初に仮想的な空間を作成し、その中にオブジェクトを配置します。2次元エンジンなら矩形、円、ポリゴンを、3次元のエンジンであれば立方体や球となるでしょう。エンジンによっては複雑な形状を指定したり、それらを組み合わせたりすることが可能です。それぞれのオブジェクトは固定されているものと動きのあるものに大別されますが、動いているものであれば速度や加速度、回転といったパラメタを指定します。

2次元の物理エンジン

壁（固定）

初期化がおわったら、この仮想世界の時間を少しだけ進めます。すると、速度の設定されているオブジェクトは新しい場所へ移動します。重力加速度が設定されている場合は、その加速度も考慮に入れます。場所の移動が完了したら画面を更新します。

オブジェクトの移動

壁（固定）

壁（固定）

この作業を繰り返し行います。するとそのうちオブジェクト同士が衝突します。衝突したらオブジェクトの向きや速度を変化させます。物理エンジンの基本的な動作はこれを繰り返すだけです。

- 時計を進めて場所を計算 ➡ 描画 ➡ 時計を進めて場所を計算 ➡ 描画 ➡ …

このように物理エンジンの原理は簡単です。面倒なのは移動や衝突時の計算です。以下のような処理を計算で求める必要があります。

- オブジェクトの位置計算（速度・加速度・摩擦・重力・回転等）
- 衝突判定
- 衝突時の処理（反発係数・重力・エネルギー保存則等）

これらの計算をどこまで正確に行うか、どの程度複雑な形状（とその組み合わせ）をサポートするか、といったところが物理エンジンの特徴につながっていきます。本書で紹介するエンジンはあくまでも入門用なので円・矩形・直線しかサポートしません。またオブジェクトの回転もサポートしません。移動するのは円のみで矩形と直線は仮想空間内で固定されているものとします。かなり割り切った仕様ですが、シンプルなゲームには十分利用できます。

6-1-3 ｜ 本書で使用した物理エンジンのソースコード

今回実装した物理エンジンTiny2D.jsの全ソースコードを次に示します。エンジンを使うだけであれば内容を理解する必要はないのでご安心ください。“こんな程度の行数で実装されているんだ”と眺めていただければ十分です。しかしながら、内容を理解できれば自分で機能を追加したり、パフォーマンスをチューニングしたりできるようになります。物理エンジンの実装を理解するにはどうしてもある程度の数学、特にベクトル演算の理解が必要となります。おそらく高校1～2年程度の数学で対応できる範囲でしょう。数学は苦手という人もいるかもしれませんが、ぜひ以下のURLにある解説記事をのぞいてみてください。

https://thinkit.co.jp/series/4770

```
"use strict";

var BodyStatic = 1;
var BodyDynamic = 2;
var ShapeCircle = 3;
var ShapeRectangle = 4;
var ShapeLine = 5;

function Vec(x, y) {
    this.x = x;
    this.y = y;
}

Vec.prototype.add = function (v) {       // 加算
    return new Vec(this.x + v.x, this.y + v.y);
}

Vec.prototype.mul = function (x, y) {   // 掛算
    var y = y || x;
    return new Vec(this.x * x, this.y * y);
}

Vec.prototype.dot = function (v) {       // 内積
    return this.x * v.x + this.y * v.y;
}

Vec.prototype.cross = function (v) {    // 外積
    return this.x * v.y - v.x * this.y;
}

Vec.prototype.move = function (dx, dy) {// 自分を移動
    this.x += dx;
    this.y += dy;
}

// 矩形オブジェクト
function RectangleEntity(x, y, width, height) {
    this.shape = ShapeRectangle;
    this.type = BodyStatic;
    this.x = x;
    this.y = y;
    this.w = width;
    this.h = height;
    this.deceleration = 1.0;
```

```
    this.isHit = function (i, j) {
        return (this.x <= i && i <= this.x + this.w &&
            this.y <= j && j <= this.y + this.h)
    }
}

// 線オブジェクト
function LineEntity(x0, y0, x1, y1, restitution) {
    this.shape = ShapeLine;
    this.type = BodyStatic;
    this.x = (x0 + x1) / 2;
    this.y = (y0 + y1) / 2;
    this.x0 = x0;
    this.y0 = y0;
    this.x1 = x1;
    this.y1 = y1;

    this.restitution = restitution || 0.9;
    this.vec = new Vec(x1 - x0, y1 - y0);
    var length = Math.sqrt(Math.pow(this.vec.x, 2) + Math.pow(this.vec.y, 2));
    this.norm = new Vec(y0 - y1, x1 - x0).mul(1 / length);
}

// 円オブジェクト
function CircleEntity(x, y, radius, type, restitution, deceleration) {
    this.shape = ShapeCircle;
    this.type = type || BodyDynamic;
    this.x = x;
    this.y = y;
    this.radius = radius;
    this.restitution = restitution || 0.9;
    this.deceleration = deceleration || 1.0;
    this.accel = new Vec(0, 0);
    this.velocity = new Vec(0, 0);

    this.move = function (dx, dy) { // 円を移動
        this.x += dx;
        this.y += dy;
    }

    this.isHit = function (x, y) {
        var d2 = Math.pow(x - this.x, 2) + Math.pow(y - this.y, 2);
        return d2 < Math.pow(this.radius, 2);
    }
```

```
    this.collidedWithRect = function (r) {  // 円と矩形の衝突
        // 矩形の4辺上で最も円に近い座標(nx, ny)を求める
        var nx = Math.max(r.x, Math.min(this.x, r.x + r.w));
        var ny = Math.max(r.y, Math.min(this.y, r.y + r.h));

        if (!this.isHit(nx, ny)) {       // 衝突なし→リターン
            return;
        }

        if (this.onhit) {                // 衝突時のコールバック
            this.onhit(this, r);
        }

        var d2 = Math.pow(nx - this.x, 2) + Math.pow(ny - this.y, 2);
        var overlap = Math.abs(this.radius - Math.sqrt(d2));
        var mx = 0, my = 0;

        if (ny == r.y) {                 // 上辺衝突
            my = -overlap;
        } else if (ny == r.y + r.h) {    // 下辺衝突
            my = overlap;
        } else if (nx == r.x) {          // 左辺衝突
            mx = -overlap;
        } else if (nx == r.x + r.w) {    // 右辺衝突
            mx = overlap;
        } else {    // 矩形の中
            mx = -this.velocity.x;
            my = -this.velocity.y;
        }

        this.move(mx, my);
        if (mx) {    // X軸方向へ反転
            this.velocity = this.velocity.mul(-1 * this.restitution, 1);
        }
        if (my) {    // Y軸方向へ反転
            this.velocity = this.velocity.mul(1, -1 * this.restitution);
        }
    }

    this.collidedWithLine = function (line) {  // 円と線の衝突
        var v0 = new Vec(line.x0 - this.x + this.velocity.x, line.y0 - this.y + this.
velocity.y);
        var v1 = this.velocity;
        var v2 = new Vec(line.x1 - line.x0, line.y1 - line.y0);
        var cv1v2 = v1.cross(v2);
```

```
        var t1 = v0.cross(v1) / cv1v2;
        var t2 = v0.cross(v2) / cv1v2;
        var crossed = (0 <= t1 && t1 <= 1) && (0 <= t2 && t2 <= 1);

        if (crossed) {
            this.move(-this.velocity.x, -this.velocity.y);
            var dot0 = this.velocity.dot(line.norm);   // 法線と速度の内積
            var vec0 = line.norm.mul(-2 * dot0);
            this.velocity = vec0.add(this.velocity);
            this.velocity = this.velocity.mul(line.restitution * this.restitution);
        }
    }

    this.collidedWithCircle = function (peer) {  // 円と円の衝突
        var d2 = Math.pow(peer.x - this.x, 2) + Math.pow(peer.y - this.y, 2);
        if (d2 >= Math.pow(this.radius + peer.radius, 2)) {
            return;
        }

        if (this.onhit) {
            this.onhit(this, peer);
        }
        if (peer.onhit) {
            peer.onhit(peer, this);
        }

        var distance = Math.sqrt(d2) || 0.01;
        var overlap = this.radius + peer.radius - distance;

        var v = new Vec(this.x - peer.x, this.y - peer.y);
        var aNormUnit = v.mul(1 / distance);         // 法線単位ベクトル1
        var bNormUnit = aNormUnit.mul(-1);           // 法線単位ベクトル2


        if (this.type == BodyDynamic && peer.type == BodyStatic) {
            this.move(aNormUnit.x * overlap, aNormUnit.y * overlap);
            var dot0 = this.velocity.dot(aNormUnit);   // 法線と速度の内積
            var vec0 = aNormUnit.mul(-2 * dot0);
            this.velocity = vec0.add(this.velocity);
            this.velocity = this.velocity.mul(this.restitution);
        }
        else if (peer.type == BodyDynamic && this.type == BodyStatic) {
            peer.move(bNormUnit.x * overlap, bNormUnit.y * overlap);
            var dot1 = peer.velocity.dot(bNormUnit);   // 法線と速度の内積
            var vec1 = bNormUnit.mul(-2 * dot1);
```

```
            peer.velocity = vec1.add(peer.velocity);
            peer.velocity = peer.velocity.mul(peer.restitution);
        }
        else {
            this.move(aNormUnit.x * overlap / 2, aNormUnit.y * overlap / 2);
            peer.move(bNormUnit.x * overlap / 2, bNormUnit.y * overlap / 2);

            var aTangUnit = new Vec(aNormUnit.y * -1, aNormUnit.x); // 接線ベクトル1
            var bTangUnit = new Vec(bNormUnit.y * -1, bNormUnit.x); // 接線ベクトル2

            var aNorm = aNormUnit.mul(aNormUnit.dot(this.velocity)); // aベクトル法線成分
            var aTang = aTangUnit.mul(aTangUnit.dot(this.velocity)); // aベクトル接線成分
            var bNorm = bNormUnit.mul(bNormUnit.dot(peer.velocity)); // bベクトル法線成分
            var bTang = bTangUnit.mul(bTangUnit.dot(peer.velocity)); // bベクトル接線成分

            this.velocity = new Vec(bNorm.x + aTang.x, bNorm.y + aTang.y);
            peer.velocity = new Vec(aNorm.x + bTang.x, aNorm.y + bTang.y);
        }
    }
}

// 物理エンジン
function Engine(x, y, width, height, gravityX, gravityY) {
    this.worldX = x || 0;
    this.worldY = y || 0;
    this.worldW = width || 1000;
    this.worldH = height || 1000;
    this.gravity = new Vec(gravityX, gravityY);
    this.entities = [];

    this.setGravity = function (x, y) {
        this.gravity.x = x;
        this.gravity.y = y;
    }

    this.step = function (elapsed) {
        var gravity = this.gravity.mul(elapsed, elapsed);
        var entities = this.entities;

        // entityを移動
        entities.forEach(function (e) {
            if (e.type == BodyDynamic) {
                var accel = e.accel.mul(elapsed, elapsed);
                e.velocity = e.velocity.add(gravity);
                e.velocity = e.velocity.add(accel);
```

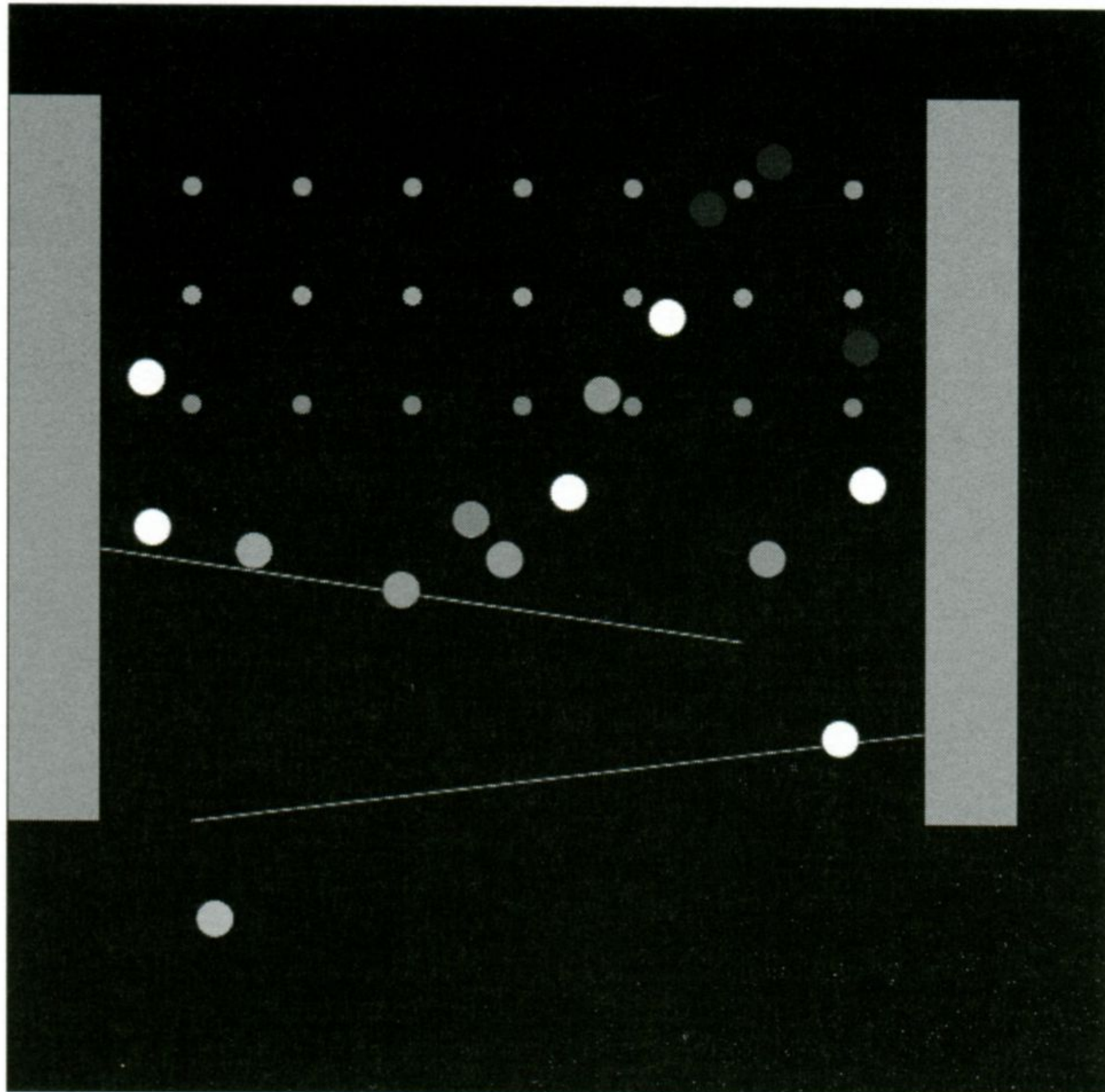
```
            e.velocity = e.velocity.mul(e.deceleration);
            e.move(e.velocity.x, e.velocity.y);
        }
    });

    // 範囲外のオブジェクトを削除
    this.entities = entities.filter(function (e) {
        return this.worldX <= e.x && e.x <= this.worldX + this.worldW &&
            this.worldY <= e.y && e.y <= this.worldY + this.worldH;
    }, this);

    // 衝突判定 & 衝突処理
    for (var i = 0 ; i < entities.length - 1 ; i++) {
        for (var j = i + 1; j < entities.length ; j++) {
            var e0 = entities[i], e1 = entities[j];
            if (e0.type == BodyStatic && e1.type == BodyStatic) {
                continue;
            }

            if (e0.shape == ShapeCircle && e1.shape == ShapeCircle) {
                e0.collidedWithCircle(e1);
            } else if (e0.shape == ShapeCircle && e1.shape == ShapeLine) {
                e0.collidedWithLine(e1);
            } else if (e0.shape == ShapeLine && e1.shape == ShapeCircle) {
                e1.collidedWithLine(e0);
            } else if (e0.shape == ShapeCircle && e1.shape == ShapeRectangle) {
                e0.collidedWithRect(e1);
            } else if (e0.shape == ShapeRectangle && e1.shape == ShapeCircle) {
                e1.collidedWithRect(e0);
            }
        }
    }
  }
}
```

6-2 物理エンジンを使ったゲーム例

いよいよ、本書の締めくくりとなる物理エンジンを使ったゲームです。ここまで読み進んでこられた方であれば問題なく理解できると思います。あとはちょっとした好奇心・想像力を働かせて、ぜひ自分のゲームを作ってみてください。

6-2-1 | デモ（demo.html）

矩形、線、円（固定）、円（移動）といったオブジェクトを画面上に配置しただけのサンプルです。シンプルなページですが、それなりにおもしろい動きをします。

demo

SAMPLE demo.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <style>
        #canvas {
            width: 600px; height: 600px;
        }
```

```
</style>
<script src="Tiny2D.js"></script>
<script>
    "use strict";
    var engine, ctx;
    var colors = ["yellow", "green", "orange", "blue", "white"];

    function rand(v) {
        return Math.floor(Math.random() * v);
    }

    function init() {    ←1
        var r;
        engine = new Engine(0, 0, 600, 800, 0, 9.8);

        r = new RectangleEntity(500, 50, 50, 400);
        r.color = "green";
        engine.entities.push(r);

        r = new RectangleEntity(0, 50, 50, 400);
        r.color = "yellow";
        engine.entities.push(r);

        r = new LineEntity(50, 300, 400, 350);
        r.color = "orange";
        engine.entities.push(r);

        r = new LineEntity(500, 400, 100, 450);
        r.color = "orange";
        engine.entities.push(r);

        for (var i = 0 ; i < 7 ; i++) {
            for (var j = 0 ; j < 3 ; j++) {
                r = new CircleEntity(i * 60 + 100, j * 60 + 100, 5, BodyStatic);
                r.color = colors[j];
                engine.entities.push(r);
            }
        }

        for (var i = 0 ; i < 20 ; i++) {
            r = new CircleEntity(rand(400)+50, rand(200), 10, BodyDynamic);
            r.color = colors[rand(5)];
            r.velocity.x = rand(10) - 5;
            r.velocity.y = rand(10) - 5;    ←2
            engine.entities.push(r);
```

```
            }

            ctx = document.getElementById("canvas").getContext("2d");
            setInterval(tick, 50);
        }

        function tick() {    ←3
            engine.step(0.01);    ←4
            repaint();
        }

        function repaint() {    ←5
            ctx.fillStyle = "black";    ←6
            ctx.fillRect(0, 0, 600, 600);
            for (var i = 0 ; i < engine.entities.length; i++) {    ←7
                var e = engine.entities[i];
                ctx.fillStyle = e.color;
                ctx.strokeStyle = e.color;
                switch (e.shape) {
                    case ShapeRectangle:
                        ctx.fillRect(e.x, e.y, e.w, e.h);
                        break;
                    case ShapeCircle:
                        ctx.beginPath();
                        ctx.arc(e.x, e.y, e.radius, 0, Math.PI * 2);
                        ctx.closePath();
                        ctx.fill();
                        break;
                    case ShapeLine:
                        ctx.beginPath();
                        ctx.moveTo(e.x0, e.y0);
                        ctx.lineTo(e.x1, e.y1);
                        ctx.stroke();
                        break;
                }
            }
        }
    </script>
</head>
<body onload="init()">
    <canvas id="canvas" width="600" height="600"></canvas>
</body>
</html>
```

物理エンジンはいろいろなページから参照するので別のファイル（「Tiny2D.js」P.283）に保存しました。外部のJavaScriptファイルを取り込む場合は以下のようにscript要素を使用します。

```
<script src="Tiny2D.js"></script>
```

ではプログラムを見ていきましょう。広域変数は、engine（物理エンジンオブジェクト）、ctx（グラフィックコンテキスト）、colors（色の配列）の3つだけです。関数rand(v)は整数の乱数を返します。

▶1 init()

init()から実行が開始されます。物理世界を作成してオブジェクトを配置しています。

```
function init() {
    var r;
    engine = new Engine(0, 0, 600, 800, 0, 9.8);     ←Ⓐ

    r = new RectangleEntity(500, 50, 50, 400);
    r.color = "green";
    engine.entities.push(r);     ←Ⓑ

    r = new RectangleEntity(0, 50, 50, 400);
    r.color = "yellow";
    engine.entities.push(r);

    r = new LineEntity(50, 300, 400, 350);
    r.color = "orange";
    engine.entities.push(r);

    r = new LineEntity(500, 400, 100, 450);
    r.color = "orange";
    engine.entities.push(r);
    for (var i = 0 ; i < 7 ; i++) {
        for (var j = 0 ; j < 3 ; j++) {
            r = new CircleEntity(i * 60 + 100, j * 60 + 100, 5, BodyStatic);
            r.color = colors[j];
            engine.entities.push(r);
        }
    }
```

物理世界はEngineオブジェクトとして実装されており、以下の命令で作成しますⒶ。

```
engine = new Engine(0, 0, 600, 800, 0, 9.8);
```

Engineオブジェクトのコンストラクタの引数は、世界のx座標、y座標、幅、高さ、x方向の重力、y方向の重力です。ここでは、左上座標(0,0)、幅600、高800で下方向に重力がある世界を作っています。

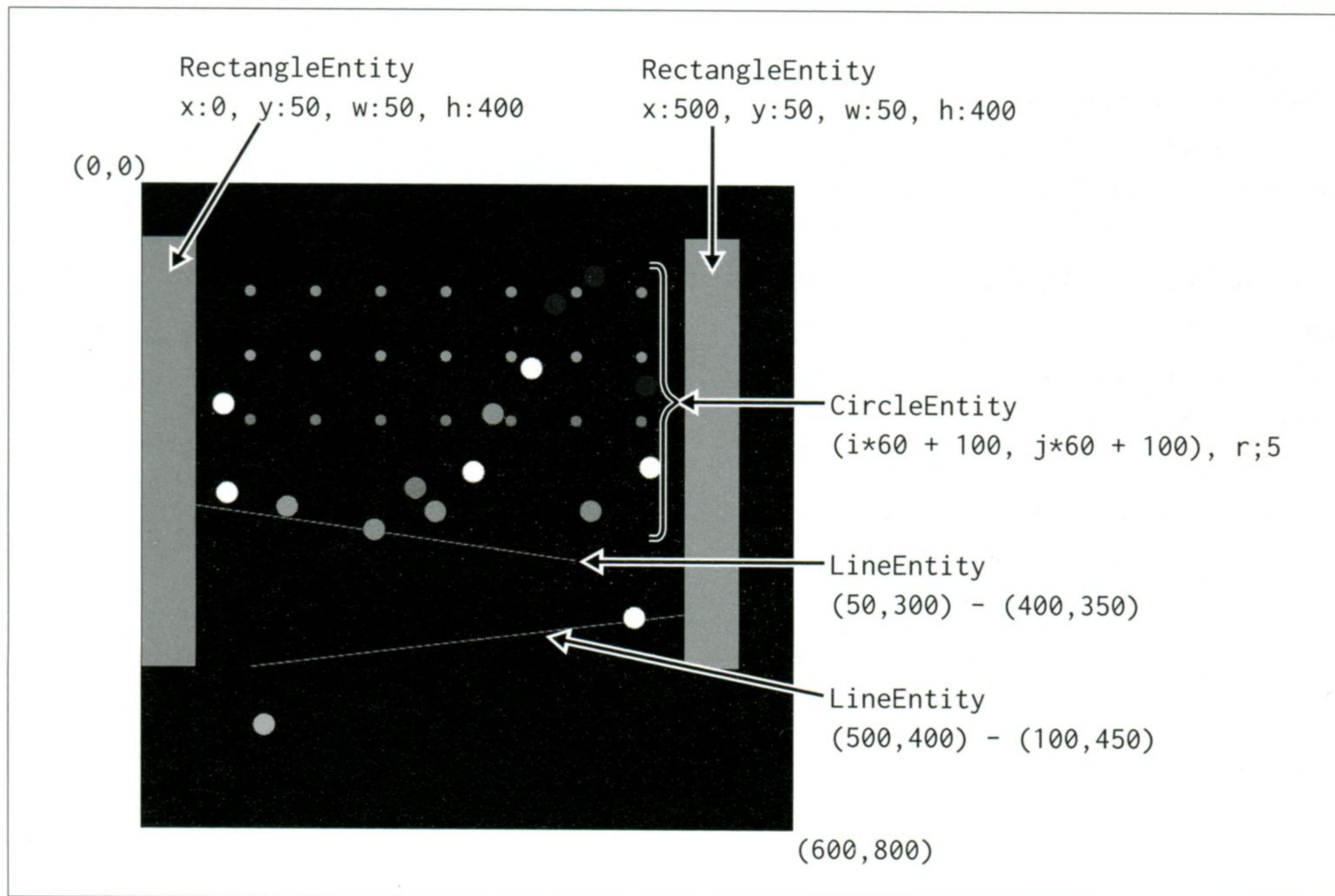
Tiny2Dでサポートしている物理オブジェクトは以下の3種類です。

- `RectangleEntity(x, y, width, height)`
  (x, y)を左上座標とする幅width、高さheightの矩形を作成します。

- `CircleEntity(x, y, radius, type, restitution, deceleration)`
  (x, y)を中心座標とする半径radiusの円を作成します。typeはBodyStatic（円が固定されている）か、BodyDynamic（動的に動くか）を指定します。デフォルトはBodyDynamicです。restitutionは反発係数、decelerationは減速度合いとなります。restitution, decelerationは省略可能です。

- `LineEntity(x0, y0, x1, y1, restitution)`
  (x0, y0)から(x1, y1)への線を引きます。restitutionは反発係数です。restitutionは省略可能です。

矩形、円、線と作成していますが、それぞれのオブジェクトのコンストラクタを呼び出してオブジェクトを作成しているだけです。とくに難しいところはないと思います。作成しているオブジェクトの様子を次の図に示します。

矩形、円、線オブジェクトの作成

作成したオブジェクトはengine.entities.push(r)で物理世界に追加しますⒷ。あとは、**4**のengine.step() で物理世界の時計を進めれば、物理世界のオブジェクトが動き始めます。その座標を取得して描画すれば、あたかも物が動いているようにみえるというわけです。

ちなみに、円オブジェクトは動かすことができますが、その初速度を**2**で

```
r.velocity.x = rand(10) - 5;
r.velocity.y = rand(10) - 5;
```

のように設定しています。あとは最後にグラフィックコンテキストを取得し、setIntervalでメインループを開始しています。

JavaScriptではオブジェクトにプロパティを追加することができます。RectangleEntityやCircleEntityといった物理世界のオブジェクトも例外ではありません。今回は描画用にcolorプロパティを追加しています。

▶3 tick()

tick()では、エンジンの時間を0.01進め、再描画を行うという処理を行っています。

▶5 repaint()

repaint()では再描画を行います。まず**6**で、画面全体を黒で塗りつぶしてクリアします。

物理世界にあるオブジェクトは物理エンジンのentitiesプロパティ（配列）に格納されているので、**7**のfor文で要素を順番に取り出します。

```
for (var i = 0 ; i < engine.entities.length; i++) {
                var e = engine.entities[i];
                ctx.fillStyle = e.color;          ←Ⓐ
                ctx.strokeStyle = e.color;
                switch (e.shape) {  ←Ⓑ
                    case ShapeRectangle:
                        ctx.fillRect(e.x, e.y, e.w, e.h);
                        break;
                    case ShapeCircle:
                        ctx.beginPath();
                        ctx.arc(e.x, e.y, e.radius, 0, Math.PI * 2);
                        ctx.closePath();
                        ctx.fill();
                        break;
```

```
                case ShapeLine:
                    ctx.beginPath();
                    ctx.moveTo(e.x0, e.y0);
                    ctx.lineTo(e.x1, e.y1);
                    ctx.stroke();
                    break;
            }
        }
```

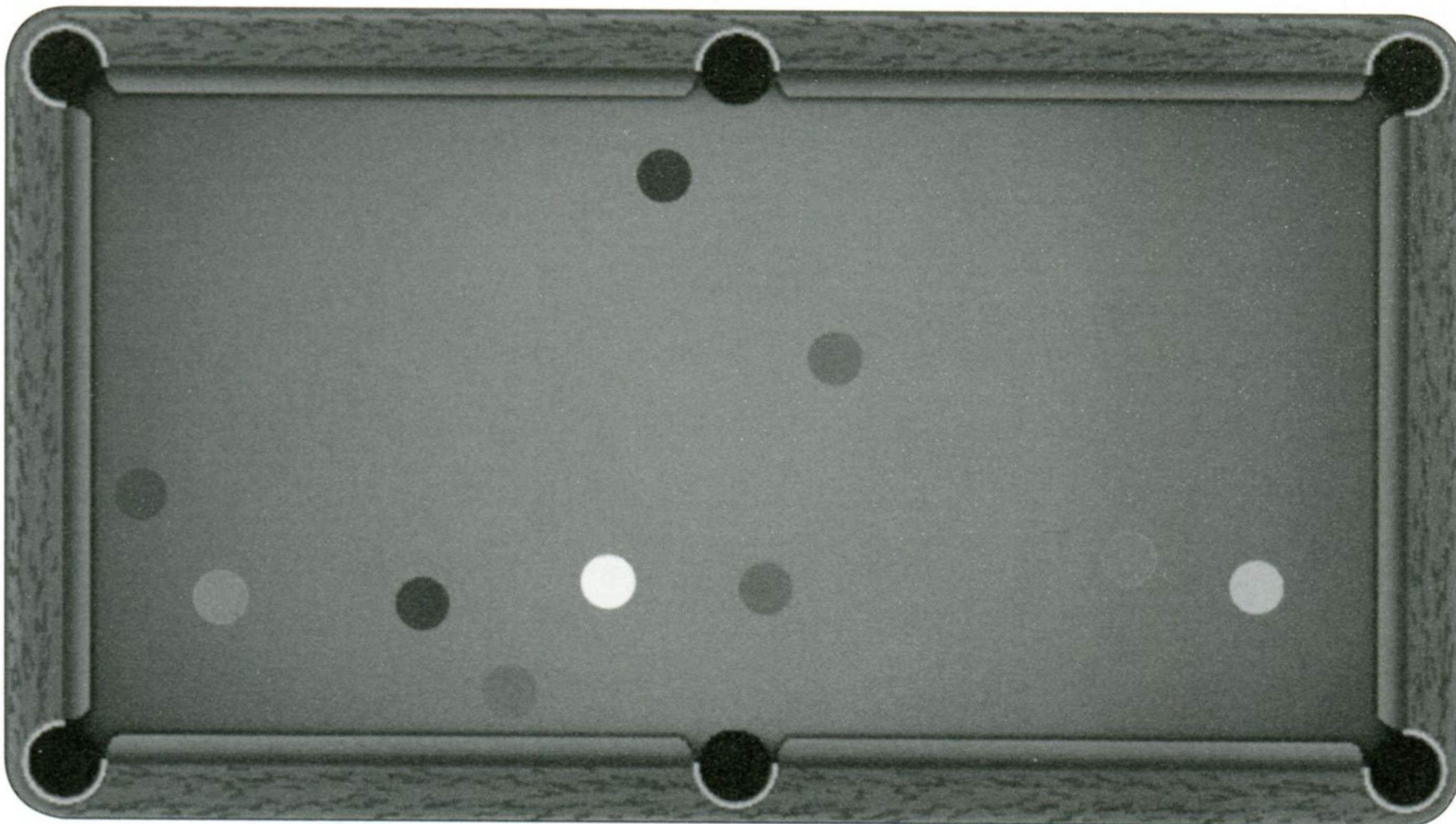
Ⓐで、コンテキストのfillStyle（塗りつぶし色）とstrokeStyle（描画色）をその要素のcolorプロパティで設定し、Ⓑのswitch文を使って、形状に応じた処理を呼び出しています。ShapeRectangle（矩形）のときはfillRectを使って矩形を描画し、ShapeCircle（円）のときはarcを使って円を描画し、ShapeLine（線）のときはmoveToとlineToで線を描画しています。

たったこれだけで物理オブジェクトが画面上を動き回ってくれるのです。おもしろいとおもいませんか?

ところで、このデモを実行していると、円が線を飛び越えるという現象に気づいた人もいると思います。実はknown issueです。動く円が何かと衝突して向きを変えるとき、その速度ベクトルを変更するとともに、めり込みを解消するため重なり量を移動させています。実はその際にも本当は衝突判定をすべきなのですが、このエンジンでは衝突判定をしていないのです。よって、移動量が大きかったりする場合に、このような現象が起きてしまいます。修正も考えたのですがコード量が増えそうだったので今回は見送りました。ご了承ください。

6-2-2 | ビリヤード（billiard.html）

手玉の上でマウスを押してドラッグし、マウスを離すとそのボールが動きだします。

ビリヤード

```
<!DOCTYPE html>
<html>
<head>
    <META charset="UTF-8">
    <style>
        #canvas {
            width: 800px; height: 450px;
            touch-action: none;
        }
    </style>
    <script src="Tiny2D.js"></script>
    <script>
        "use strict";
        var ctx, engine, target, mousePos = null;

        var walls = [
            [-100, -100, 1000, 140],
            [-100, 410, 1000, 100],
            [-100, -100, 140, 650],
            [760, -100, 100, 650],
        ];
        var holes = [
            [35, 35], [400, 35], [765, 35], [35, 415], [400, 415], [765, 415],
        ];
        var balls = [
            { x: 200, y: 200, c: "#FFF400" },
            { x: 125, y: 185, c: "#005CD3" },
            { x: 150, y: 170, c: "#CE2721" },
            { x: 100, y: 200, c: "#BD4CB8" },
            { x: 175, y: 215, c: "#F06700" },
            { x: 125, y: 215, c: "#0B8A17" },
            { x: 175, y: 185, c: "#B70D3A" },
            { x: 150, y: 230, c: "#333333" },
            { x: 150, y: 200, c: "#FFD300" },
            { x: 650, y: 200, c: "#CAFDFF" },
        ];

        function init() {    ←1
            // エンジン初期化 & イベントハンドラ設定
            engine = new Engine(-100, -100, 1000, 650, 0, 0);
            var canvas = document.getElementById("canvas");
            canvas.onmousedown = mymousedown;
            canvas.onmousemove = mymousemove;
            canvas.onmouseup = mymouseup;
```

```
    canvas.addEventListener('touchstart', mymousedown);
    canvas.addEventListener('touchmove', mymousemove);
    canvas.addEventListener('touchend', mymouseup);

    // 壁
    walls.forEach(function (w) {
        var r = new RectangleEntity(w[0], w[1], w[2], w[3]);
        r.color = "gray";                                              ←2
        engine.entities.push(r);
    });

    // ボール
    balls.forEach(function (b) {
        var r = new CircleEntity(b.x, b.y, 15, BodyDynamic, 0.9, 0.99);
        r.color = b.c;
        b.entity = r;                                                  ←3
        engine.entities.push(r);
    });

    holes.forEach(function (h) {
        var r = new CircleEntity(h[0], h[1], 20, BodyStatic);
        r.color = "rgba(255,255,255,0)";
        r.onhit = function (me, peer) {
            engine.entities = engine.entities.filter(function (e) {
                return e != peer;                                      ←4
            });
        }
        engine.entities.push(r);
    });

    // その他(Canvas, Timer)の初期化
    ctx = canvas.getContext("2d");
    ctx.font = "20pt Arial";
    ctx.strokeStyle = "blue";
    setInterval(tick, 50);
}

function tick() {
    engine.step(0.01);  // 物理エンジンの時刻を進める
    repaint();          // 再描画
}

function mymousedown(e) {          ←5
    var mouseX = !isNaN(e.offsetX) ? e.offsetX : e.touches[0].clientX;
    var mouseY = !isNaN(e.offsetY) ? e.offsetY : e.touches[0].clientY;   ←6
```

```
        for (var i = 0 ; i < balls.length ; i++) {
            if (balls[i].entity.isHit(mouseX, mouseY)) {      ←7
                target = balls[i].entity;
                mousePos = { x: mouseX, y: mouseY };
                break;
            }
        }
    }

    function mymousemove(e) {      ←8
        var mouseX = !isNaN(e.offsetX) ? e.offsetX : e.touches[0].clientX;
        var mouseY = !isNaN(e.offsetY) ? e.offsetY : e.touches[0].clientY;
        if (target) {
            mousePos = { x: mouseX, y: mouseY };
        }
    }

    function mymouseup(e) {      ←9
        if (target) {
            var dx = mousePos.x - target.x;
            var dy = mousePos.y - target.y;
            target.velocity.x = dx / 10;
            target.velocity.y = dy / 10;
        }
        target = null;
    }

    function repaint() {
        // 背景クリア
        ctx.drawImage(billiard, 0, 0, 800, 450);

        // ボール・壁の描画
        for (var i = 0 ; i < engine.entities.length; i++) {
            var e = engine.entities[i];
            ctx.fillStyle = e.color;
            switch (e.shape) {
                case ShapeCircle:
                    ctx.beginPath();
                    ctx.arc(e.x, e.y, e.radius, 0, Math.PI * 2);
                    ctx.closePath();
                    ctx.fill();
                    break;
            }
        }
```

```
            if (target && mousePos) {
                ctx.beginPath();
                ctx.moveTo(target.x, target.y);
                ctx.lineTo(mousePos.x, mousePos.y);
                ctx.stroke();
            }
        }
    </script>
</head>
<body onload="init()">
    <canvas id="canvas" width="800" height="450"></canvas>
    <img id="billiard" src="billiard.png" style="display:none"/>
</body>
</html>
```

基本的なつくりはdemo.htmlと同じです。説明が重複する部分は割愛します。広域変数のwallsは壁、holesは穴、ballsは玉の座標情報です。

▶1 init()

初期化を行います。まず各種イベントハンドラを設定します。次に2で、wallsの要素をforEach文を使って順番にとりだし、RectangleEntityオブジェクトを作って壁を構築しています。Array.forEachメソッドを使っていますがfor文でもかまいません。好きなほうを使ってください。

ボールも同様にオブジェクトを作成しています3。穴だけは衝突処理が必要なので若干異なります。穴にボールがぶつかったときは（=穴にボールが落ちた）ボールを削除する必要があるからです。4のコードでその処理を行っています。

```
r.onhit = function (me, peer) {
    engine.entities = engine.entities.filter(function (e) {
        return e != peer;
    });
}
```

rは穴のオブジェクトです。これにonhitメソッドを追加しています。このメソッドは自身がほかのオブジェクトと衝突したときに呼び出されます。引数のmeは自分で、peerは衝突相手のオブジェクトです。

Array.filterを使いpeer以外の要素、すなわち衝突相手以外の要素を抽出しています。filterの引数には関数を指定しますが、配列内の要素がpeerでないとき（「e != peer」）にtrueを返しています。こうすることで、衝突した要素のみがエンジンのentitiesから取り除かれ、物理世界からなくなります。あとは、グラフィックコンテキストを取得しメインループを開始しているだけです。

▶5 mymousedown(e)

マウス押下時のコールバック関数です。玉の上でマウスが押されたときは、その玉を手玉として覚えておき、マウスが移動した場所までの線を描画します。マウスが離されたときに、その距離と向きをもとに手玉に初速度を与えます。

マウス押下時の座標は「(e.offsetX, e.offsetY)」で取得できます。タッチの場合は「e.touches[0].clientX」で取得します6。

この座標が玉に含まれているかを7の「balls[i].entity.isHit(mouseX, mouseY)」で判定しています。ここで「balls[i].isHit(mouseX, mouseY)」ではないことに注意してください。balls[i]は物理世界の玉オブジェクトではなく、物理世界の玉を作成するための座標データを格納している広域変数だからです。init()では3のように玉オブジェクトを初期化しました。

```
balls.forEach(function (b) {
    var r = new CircleEntity(b.x, b.y, 15, BodyDynamic, 0.9, 0.99);
    r.color = b.c;
    b.entity = r;
```

rはCircleEntityオブジェクトで物理世界の玉に該当します。forEachでは配列の個々の要素が関数の引数として渡されます。上記のコードではbがそれにあたります。そのentityプロパティに物理世界のオブジェクトを格納しているので、物理世界の玉オブジェクトはballs[i].entityで参照することができるのです。ともあれ、マウスを含む円が見つかった場合、その円を手玉として広域変数targetに格納し、マウス押下時の座標をmousePosに保存しておきます。

▶8 mymousemove(e)

マウス移動時のコールバック関数です。手玉があるときはmousePosを更新します。

▶9 mymouseup(e)

マウスリリース時のコールバック関数です。手玉があるときは現在のマウス位置mousePosと手玉targetの座標の差分から手玉に初速度を与えます。

再描画の関数repaint()はdemo.htmlのコードとほとんど同じです。手玉targetとmousePosがあるときはそれらを結ぶ線を描画しています。

▶タッチデバイス対応について

このゲームではクリックされた座標が円の中にあるときにその球を手玉としています。マウスの場合、接点が1×1ピクセルなので円の中をクリックするのは簡単です。

一方、タッチデバイスの場合、指が接する面積が広くなるため円の中をクリックするのが難しくなります。そ

のため、スマホのように画面が小さいデバイスでは手玉を操作しづらくなってしまいます。

ボールを作成しているコードは以下の箇所です。

```
var r = new CircleEntity(b.x, b.y, 15, BodyDynamic, 0.9, 0.99);
```

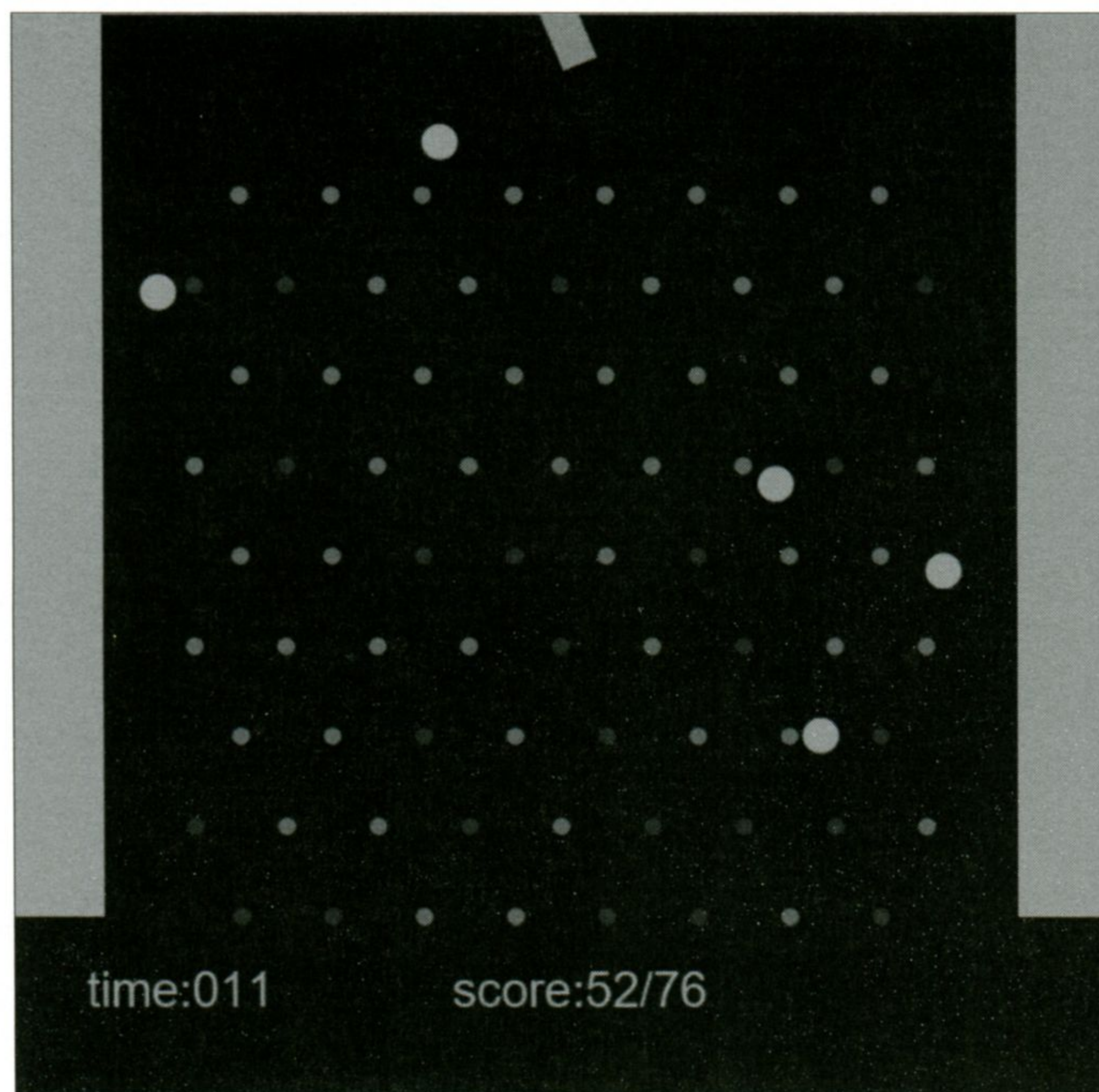
この半径15を大きくすることで玉が大きくなります。玉が大きくなれば操作性は改善するかもしれません。

ただし、玉の大きさを大きくする手法は場当たり的な対処にすぎません。スマホを想定してゲームを作る場合は、最初から指でも操作しやすい仕様にしておくことが大切です。

6-2-3 ペグ（Peg.html）

マウスをクリックした方向へ玉が発射されます。釘にあたると釘の色が変わります。すべての釘の色を変えるまでの時間を競うゲームです。このゲームを理解するために必要な知識は、本書のこれまでの説明でカバーしているので、ソースコードの説明は省略します。自分でソースコードを読んで理解するという作業をお楽しみください。

ペグ

time:011 score:52/76

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <style>
        #canvas {
            width: 600px; height: 600px;
            touch-action: none;
        }
    </style>
    <script src="Tiny2D.js"></script>
    <script>
        "use strict";
        var ctx, engine, timer, startTime = 0;
        var sound, score = 0, dir = Math.PI / 2;

        function rand(v) {
            return Math.floor(Math.random() * v);
        }

        function init() {
            // エンジン初期化 & イベントハンドラ設定
            engine = new Engine(-100, -100, 800, 800, 0, 9.8);
            var canvas = document.getElementById("canvas");
            canvas.onmousemove = mymousemove;
            canvas.onclick = myclick;
            sound = new Audio("sound0.mp3");

            // 壁
            r = new RectangleEntity(-50, 0, 100, 500);
            r.color = "yellow";
            engine.entities.push(r);

            r = new RectangleEntity(550, 0, 100, 500);
            r.color = "yellow";
            engine.entities.push(r);

            // 釘
            for (var i = 0 ; i < 9 ; i++) {
                for (var j = 0 ; j < 8 + i % 2 ; j++) {
                    var x = (j * 50 + 125) - 25 * (i % 2);
                    var r = new CircleEntity(x, i * 50 + 100, 5, BodyStatic);
                    r.onhit = function (me, peer) {
                        if (me.color == "blue") {
```

```
                    me.color = "red";
                    score++;
                    sound.pause();
                    sound.currentTime = 0;
                    sound.play();
                    if (score >= 76) {
                        clearInterval(timer);
                        timer = NaN;
                        repaint();
                    }
                }
            }
            r.color = "blue";
            engine.entities.push(r);
        }
    }

    // その他(Canvas, Timer)の初期化
    ctx = canvas.getContext("2d");
    ctx.font = "20pt Arial";
    startTime = new Date();
    timer = setInterval(tick, 50);
}

function tick() {
    engine.step(0.01);  // 物理エンジンの時刻を進める
    repaint();          // 再描画
}

function myclick(e) {
    var ball = new CircleEntity(300, 10, 10, BodyDynamic, 0.9);
    ball.velocity.x = 10 * Math.cos(dir);
    ball.velocity.y = 10 * Math.sin(dir);
    ball.color = "yellow";
    engine.entities.push(ball);
}

function mymousemove(e) {
    dir = Math.atan2(e.y, (e.x - 300));
}

function repaint() {
    // 背景クリア
    ctx.fillStyle = "black";
    ctx.fillRect(0, 0, 600, 600);
```

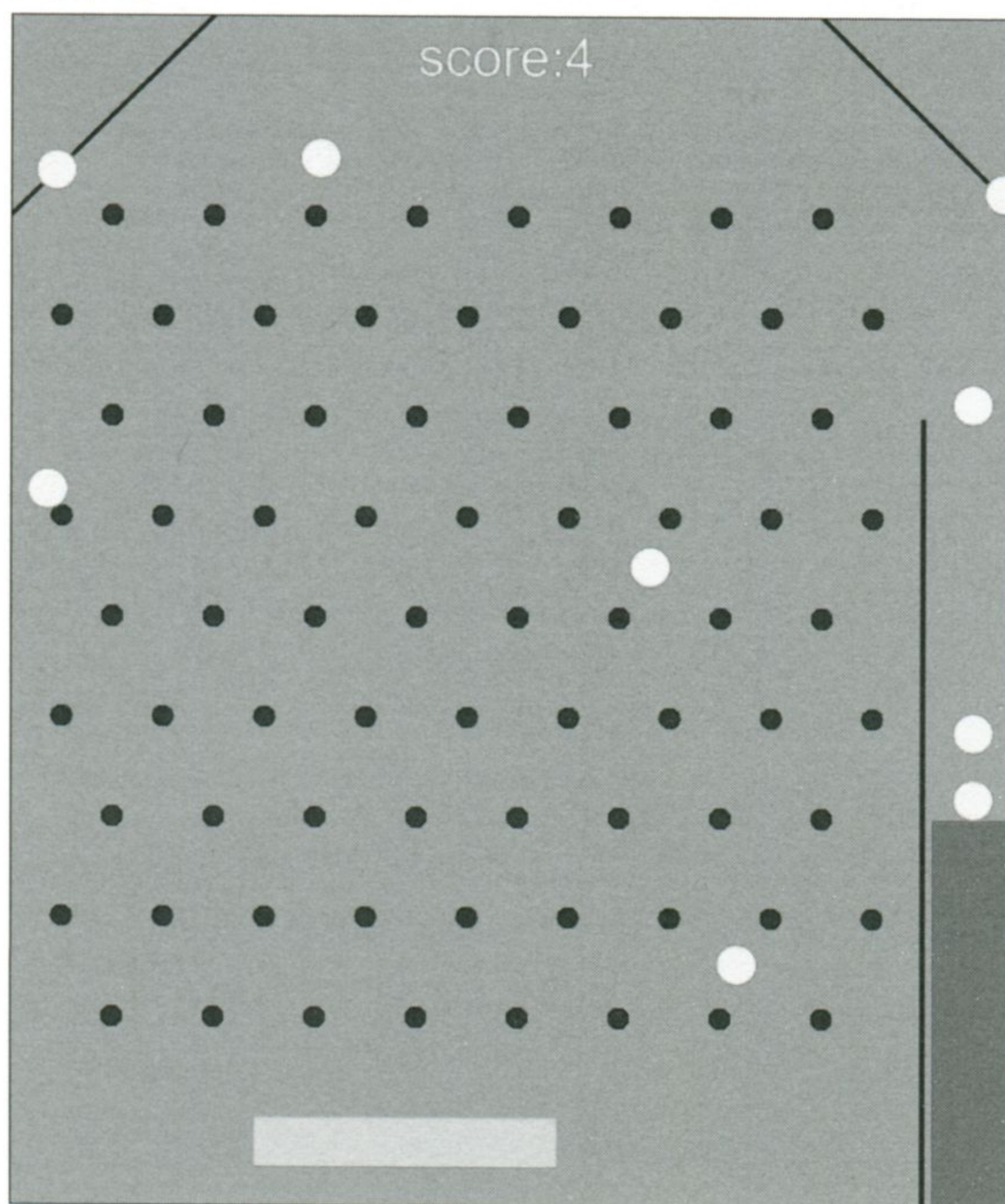
```
            // 発射台描画
            ctx.fillStyle = "orange";
            ctx.save();
            ctx.translate(300, 0);
            ctx.rotate(dir);
            ctx.fillRect(-20, -10, 50, 20);
            ctx.restore();

            // ボール・壁・釘の描画
            for (var i = 0 ; i < engine.entities.length; i++) {
                var e = engine.entities[i];
                ctx.fillStyle = e.color;
                switch (e.shape) {
                    case ShapeCircle:
                        ctx.beginPath();
                        ctx.arc(e.x, e.y, e.radius, 0, Math.PI * 2);
                        ctx.closePath();
                        ctx.fill();
                        break;
                    case ShapeRectangle:
                        ctx.fillRect(e.x, e.y, e.w, e.h);
                        break;
                }
            }

            // 各種情報表示
            var elapsed = Math.floor((new Date().getTime() - startTime) / 1000);
            ctx.fillText("score:" + score + "/76", 240, 550);
            ctx.fillText("time:" + ('000' + elapsed).slice(-3), 40, 550);
            if (isNaN(timer)) {
                ctx.fillText("CLEARED!!", 220, 300);
            }
        }
    </script>
</head>
<body onload="init()">
    <canvas id="canvas" width="600" height="600"></canvas>
</body>
</html>
```

mymousemove(e)については補足しておきましょう。Math.atan2(y, x)はアークタンジェントといって、座標から角度（ラジアン）を求める関数です。引数の順番がy座標、x座標となっていることに注意してください。求めた角度を広域変数dirに格納し、砲台の描画や玉の発射角の計算に使用しています。

また、効果音再生時に「sound.currentTime = 0」と再生位置を設定しています。今回の効果音は最初にアタック部分があり、1秒程度かけて減衰していきます。普通に再生すると減衰を待ってから、次の音声が再生されるので、連続して玉が釘にぶつかった場合に適切な効果が得られません。そこで、currentTimeを設定し、より自然な効果が得られるようにしています。

6-2-4 ｜ パチンコ（Pachinko.html）

マウスを押す、もしくはタッチをするとレバーが引かれ、離すと玉が発射します。画面下部を左右に移動するターゲットに玉が当たるとスコアが加算されます。

パチンコ

score:4

```
<!DOCTYPE html>
<html>
<head>
    <title>Pachinko</title>
    <meta charset="UTF-8">
    <style>
        #canvas {
            width: 500px; height: 600px;
            touch-action: none;
        }

    </style>
    <script src="Tiny2D.js"></script>
    <script>
        "use strict";
        var ctx, engine, offset = 0, catcher, score = 0, isMouseDown = false;

        var walls = [
            [-100, -100, 100, 800],
            [-100, -100, 800, 100],
            [500, -100, 100, 800],
        ];

        var lines = [
            [150, -50, -50, 150],
            [350, -50, 550, 150],
            [450, 200, 450, 800]
        ];

        function init() {
            // エンジン初期化 & イベントハンドラ設定
            engine = new Engine(-100, -100, 700, 800, 0, 9.8);
            var canvas = document.getElementById("canvas");
            canvas.onmousedown = mymousedown;
            canvas.onmouseup = mymouseup;
            canvas.addEventListener('touchstart', mymousedown);
            canvas.addEventListener('touchend', mymouseup);
            canvas.oncontextmenu = function (e) { e.preventDefault(); };

            // 壁
            walls.forEach(function (w) {
                r = new RectangleEntity(w[0], w[1], w[2], w[3]);
                r.color = "gray";
                engine.entities.push(r);
```

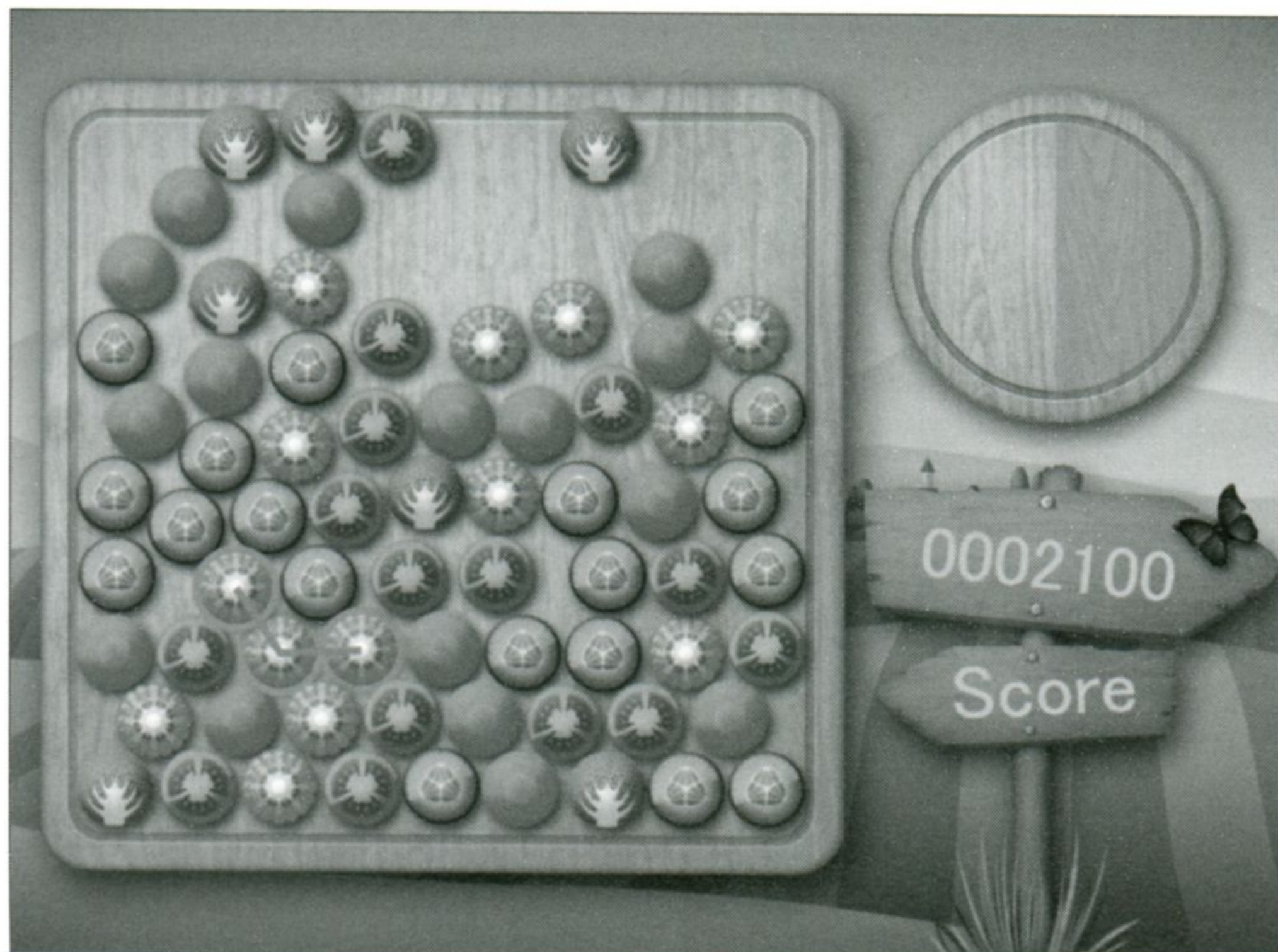
```
    });

    lines.forEach(function (w) {
        r = new LineEntity(w[0], w[1], w[2], w[3], 0.8);
        r.color = "gray";
        engine.entities.push(r);
    });

    // 釘
    for (var i = 0 ; i < 9 ; i++) {
        for (var j = 0 ; j < 8 + i % 2 ; j++) {
            var x = (j * 50 + 50) - 25 * (i % 2);
            var r = new CircleEntity(x, i * 50 + 100, 5, BodyStatic, 1);
            r.color = "blue";
            engine.entities.push(r);
        }
    }

    catcher = new RectangleEntity(0, 550, 150, 25);
    catcher.color = "gold";
    catcher.sign = 1;

    engine.entities.push(catcher);

    // その他(Canvas, Timer)の初期化
    ctx = canvas.getContext("2d");
    ctx.font = "20pt Arial";
    ctx.strokeStyle = "blue";
    timer = setInterval(tick, 50);
}

function tick() {
    if (isMouseDown) {
        offset = Math.min(offset + 5, 200);
    }
    catcher.sign *= (catcher.x > 300 || catcher.x < 0) ? -1 : 1;
    catcher.x = catcher.x + 5 * catcher.sign;

    engine.step(0.01);  // 物理エンジンの時刻を進める
    repaint();          // 再描画
}

function mymousedown(e) {
    isMouseDown = true;
}
```

```
function mymouseup(e) {
    isMouseDown = false;
    var r = new CircleEntity(475, 400, 10, BodyDynamic);
    r.color = "yellow";
    r.velocity.y = -offset / 5;
    r.onhit = function (me, peer) {
        if (peer == catcher) {
            engine.entities = engine.entities.filter(function (e) {
                return e != me;
            });
            score++;
        }
    }

    offset = 0;
    engine.entities.push(r);
}

function repaint() {
    // 背景クリア
    ctx.fillStyle = "#006600";
    ctx.fillRect(0, 0, 500, 600);

    // ボール・壁の描画
    for (var i = 0 ; i < engine.entities.length; i++) {
        var e = engine.entities[i];
        ctx.fillStyle = e.color;
        switch (e.shape) {
            case ShapeCircle:
                ctx.beginPath();
                ctx.arc(e.x, e.y, e.radius, 0, Math.PI * 2);
                ctx.closePath();
                ctx.fill();
                break;
            case ShapeRectangle:
                ctx.fillRect(e.x, e.y, e.w, e.h);
                break;
            case ShapeLine:
                ctx.beginPath();
                ctx.moveTo(e.x0, e.y0);
                ctx.lineTo(e.x1, e.y1);
                ctx.stroke();
                break;
        }
```

```
            }

            ctx.fillText("score:" + score, 200, 30);
            ctx.fillStyle = "yellow";
            ctx.beginPath();
            ctx.arc(475, 390 + offset, 10, 0, Math.PI * 2);
            ctx.closePath();
            ctx.fill();

            ctx.fillStyle = "gray";
            ctx.fillRect(455, 400 + offset, 40, 200);
        }
    </script>
</head>
<body onload="init()">
    <canvas id="canvas" width="500" height="600"></canvas>
</body>
</html>
```


6-2-5 | ベジタブルマーチ（VegetableMarch.html）

この本の最後を締めくくるゲームです。近くにある同じ野菜をつなげてどんどん消していくゲームです。物理エンジンと落ちモノ系ゲームの組み合わせといったところでしょうか。一度に多くの野菜を消すほど高得点となります。

ベジタブルマーチ

```
<!DOCTYPE html>
<html>
<head>
    <title>VegetableMarch</title>
    <META charset="UTF-8">
    <style>
        #canvas {
            width: 800px;
            height: 600px;
            touch-action: none;
        }
        #START {
            position: absolute;
            left: 200px;
            top: 200px;
        }
    </style>
    <script src="Tiny2D.js"></script>
    <script>
        "use strict";

        var ctx, engine, veges = [], images = [];
        var timer = NaN, startTime = NaN, elapsed = 0, score = 0;
        var walls = [
            [-60, -100, 100, 800],
            [500, -100, 100, 800],
            [-60, 520, 700, 100],
        ];

        function rand(v) {
            return Math.floor(Math.random() * v);
        }

        function init() {        ←1
            // エンジン初期化 & Canvas初期化
            var canvas = document.getElementById("canvas");
            ctx = canvas.getContext("2d");
            ctx.font = "20pt Arial";
            ctx.strokeStyle = "blue";
            ctx.lineWidth = 5;
            ctx.textAlign = "center";

            engine = new Engine(-100, -100, 700, 700, 0, 9.8);
```

```
        // 壁
        walls.forEach(function (w) {
            var r = new RectangleEntity(w[0], w[1], w[2], w[3]);
            r.color = "gray";
            engine.entities.push(r);
        });

        // 野菜
        for (var i = 0 ; i < 7 ; i++) {
            for (var j = 0 ; j < 10 ; j++) {
                var x = i * 60 + 75 + rand(5);
                var y = j * 50 + 50 + rand(5);
                var r = new CircleEntity(x, y, 25, BodyDynamic, 1, 0.98);
                r.color = rand(5);
                engine.entities.push(r);
            }
        }

        for (var i = 0 ; i < 5 ; i++) {
            images.push(document.getElementById("fruit" + i));
        }

        repaint();
    }

    function go() {        ←2
        var canvas = document.getElementById("canvas");
        canvas.onmousedown = mymousedown;
        canvas.onmousemove = mymousemove;
        canvas.onmouseup = mymouseup;
        canvas.addEventListener('touchstart', mymousedown);
        canvas.addEventListener('touchmove', mymousemove);
        canvas.addEventListener('touchend', mymouseup);

        document.body.addEventListener('touchmove', function (event) {
            event.preventDefault();                                      ←3
        }, false);
        document.getElementById("START").style.display = "none";      ←4
        document.getElementById("bgm").play();

        startTime = new Date();
        timer = setInterval(tick, 50);
    }

    /**
```

```
 * メインループ
 */
function tick() {     ←5
    engine.step(0.01);  // 物理エンジンの時刻を進める  ←6

    elapsed = ((new Date()).getTime() - startTime) / 1000;
    if (elapsed > 57) {
        clearInterval(timer);
        timer = NaN;
    }
    repaint();          // 再描画
}

function mymousedown(evt) {       ←7
    var x = !isNaN(evt.offsetX) ? evt.offsetX : evt.touches[0].clientX;
    var y = !isNaN(evt.offsetY) ? evt.offsetY : evt.touches[0].clientY;
    engine.entities.forEach(function (e) {
        if (e.isHit(x, y) && e.shape == ShapeCircle) {
            veges.push(e);
            e.selected = true;
        }
    });
}

function mymousemove(evt) {       ←8
    if (veges.length == 0) {
        return;
    }

    var x = !isNaN(evt.offsetX) ? evt.offsetX : evt.touches[0].clientX;
    var y = !isNaN(evt.offsetY) ? evt.offsetY : evt.touches[0].clientY;
    var p = veges[veges.length - 1];

    engine.entities.forEach(function (e) {
        if (e.isHit(x, y) && e.shape == ShapeCircle) {
            if (veges.indexOf(e) < 0 && e.color == p.color) {
                var d2 = Math.pow(e.x - p.x, 2) + Math.pow(e.y - p.y, 2);
                if (d2 < 4000) {
                    veges.push(e);
                    e.selected = true;
                }
            }
        }
    });
}
```

```
function mymouseup(evt) {        ←9
    if (veges.length > 1) {
        // 選択状態の野菜を削除
        engine.entities = engine.entities.filter(function (e) {
            return e.selected != true;
        })

        // 消去分を追加
        for (var i = 0 ; i < veges.length ; i++) {
            var x = 75 + rand(350);
            var r = new CircleEntity(x, 0, 25, BodyDynamic, 1, 0.98);
            r.color = rand(5);
            engine.entities.push(r);
        }
        score += veges.length * 100;
    }
    veges.forEach(function (e) { delete e.selected })
    veges = [];
}

function repaint() {        ←10
    // 背景クリア
    ctx.drawImage(fruitbg, 0, 0);

    // 野菜
    for (var i = 0 ; i < engine.entities.length; i++) {
        var e = engine.entities[i];
        var img = images[e.color];
        if (e.shape == ShapeCircle) {
            ctx.drawImage(img, e.x - 28, e.y - 28, 62, 62);
            if (e.selected) {
                ctx.strokeStyle = "yellow";
                ctx.beginPath();
                ctx.arc(e.x, e.y, e.radius, 0, Math.PI * 2);
                ctx.closePath();
                ctx.stroke();
            }
        }
    }

    // 線
    if (veges.length > 0) {
        ctx.strokeStyle = "#B1EB22";
        ctx.beginPath()
```

```
            ctx.moveTo(veges[0].x, veges[0].y);
            for (var i = 1 ; i < veges.length ; i++) {
                ctx.lineTo(veges[i].x, veges[i].y);
            }
            ctx.stroke();
        }

        // メッセージ
        ctx.save();
        ctx.fillStyle = "#F9D79F";
        ctx.font = "bold 24pt sans-serif";
        ctx.font
        ctx.translate(650, 442);
        ctx.rotate(-0.05);
        ctx.fillText(isNaN(timer) ? "FINISH" : "Score", 0, 0);
        ctx.restore();

        // スコア
        ctx.save();
        ctx.font = "bold 32pt sans-serif";
        ctx.translate(650, 365);
        ctx.rotate(0.08);
        ctx.fillStyle = "#F9D79F";
        ctx.fillText(('0000000' + score).slice(-7), 0, 0);
        ctx.restore();

        // 残り時間
        ctx.save();
        ctx.fillStyle = 'rgba(215, 130, 40, 0.5)';
        ctx.beginPath();
        ctx.moveTo(656, 153);
        ctx.arc(656, 153, 88, -Math.PI / 2, elapsed / 57 * Math.PI * 2 - Math.PI / 2);
        ctx.closePath();
        ctx.fill();
        ctx.restore();
    }
  </script>
</head>
<body onload="init()">
  <!-- Thanks to http://takao-suenobu.com/  & http://dova-s.jp/ -->
  <audio src="bgm.mp3" id="bgm"></audio>
  <canvas id="canvas" width="800" height="600"></canvas>
  <img id="START" src="start.png" onclick="go()">
  <img id="fruitbg" src="fruitbg.png" style="display:none" />
  <img id="fruit0" src="fruit0.png" style="display:none" />
```

```
    <img id="fruit1" src="fruit1.png" style="display:none" />
    <img id="fruit2" src="fruit2.png" style="display:none" />
    <img id="fruit3" src="fruit3.png" style="display:none" />
    <img id="fruit4" src="fruit4.png" style="display:none" />
</body>
</html>
```


▶広域変数

使用している広域変数は以下のとおりです。

使用している広域変数

変数	説明
ctx	グラフィックコンテキスト
engine	物理エンジン
veges	野菜を格納する配列
timer	タイマー
startTime	ゲーム開始時刻
elapsed	経過時間
score	スコア
walls	壁オブジェクト
images	野菜の画像を格納する配列

主な関数について以下に説明します。

▶1 init()

初期化関数です。canvasのコンテキスト設定、物理エンジンオブジェクトの作成、壁や野菜オブジェクトの作成を行い、描画を明示的に行うためにrepaint()を呼び出します。ゲーム開始のタイマーを開始するのはSTARTボタンが押下されたタイミングであり、文書をロードした時点ではないことに注意してください。

▶2 go()

ゲームを開始時に呼ばれる関数です。スタートボタンの画像「<img id="START" src="start.png" onclick="go()">」のonclick属性から呼び出されます。

まず、canvasのタッチやマウスのイベントハンドラを登録します。3のコードは、タッチ時にコンテキストメニューが表示されるのを防止するための処理です。

```
document.body.addEventListener('touchmove', function (event) {
    event.preventDefault();
}, false);
```

また、4の

```
document.getElementById("START").style.display = "none";
```

でスタートボタンを非表示にしています。その後BGMの再生を開始し、setIntervalでメインループを開始しています。

▶5 tick()

6のengine.step(0.01)で物理エンジンの時刻を進めます。経過秒数を以下で求め、

```
elapsed = ((new Date()).getTime() - startTime) / 1000;
```

57秒（BGMの再生時間）を過ぎたらタイマーを停止させています。

▶7 mymousedown(evt)

マウスやタッチ押下時、その座標に野菜オブジェクトがあったならば、そのオブジェクトを配列vegesに格納し、そのオブジェクトのselectedプロパティをtrueに設定します。

▶8 mymousemove(evt)

マウス移動時のイベントハンドラです。veges配列が空のときは単にreturnします。

「var p = veges[veges.length - 1] 」では配列にある最後の野菜を取得し、変数pに格納しています。これはマウスが動いたとき、最後に選ばれた野菜と今のマウス位置にある野菜を比較し、距離が近く同じ野菜のときに限り、その野菜を選択するという処理を行う必要があるためです。その処理を行っているのが以下の部分です。

```
if (e.isHit(x, y) && e.shape == ShapeCircle) {
    if (veges.indexOf(e) < 0 && e.color == p.color) {
        var d2 = Math.pow(e.x - p.x, 2) + Math.pow(e.y - p.y, 2);
        if (d2 < 4000) {
            veges.push(e);
            e.selected = true;
        }
    }
}
```

最初のif文が、対象となるオブジェクトにマウスの座標が含まれているか、形状が円か否かを判断しています。それらの条件を満たしている場合は、次のif文ですでに野菜が選択されていないか（配列vegesに含まれていないか＝「veges.indexOf(e) ＜ 0」)、かつ、最後の野菜と今の野菜が同じ種類か（「e.color == p.color」）を判断しています。それらの条件を満たすと、最後のif文に進みます。ここで2つの野菜の距離が一定値以下であるか判断し、その条件を満たしたときに、その野菜を配列vegesに格納し、プロパティselectedをtrueに設定します。

▶9 mymouseup(evt)

マウスが離されたときのコールバックです。選択状態の野菜が2つ以上ある場合（veges.length ＞ 1)、その野菜をArray.filterを使って物理世界から取り除きます。Array.filterは条件に合致した要素だけを含む配列を返します。条件は関数メソッドで指定します。大丈夫ですよね? 消去したらその分の野菜を追加しています。これも今まで何度も見てきたコードです。

そして、選択された野菜の数に応じてスコアを加算します。最後にすべての野菜のselectedプロパティを削除し、配列vegesを空で初期化しています。

▶10 repaint()

とくに難しい処理はありません。これまでと同様に、背景をクリアして物理世界に含まれるShapeCircleを描画します。選択状態にある場合（e.selectedがtrueのとき）は円を強調描画して選択されていることが視覚的にわかるようにしています。あとは、選択状態にある円の中心を線で結んで描画し、ゲーム終了時のメッセージ、スコアの描画、残り時間の扇形の描画を行います。

一見すると複雑そうに見えるゲームですが非常にシンプルに実装できていることがわかると思います。コードの内容はビリヤード（P.295）とあまり変わりません。しかしながらアイディア次第でまったく別のゲームになっているのです。物理エンジンの威力を感じていただける一例ではないでしょうか?

索　引

記号・数字

◉記号

◉数字

アルファベット

索　引

◉N

◉O

◉P

◉R

◉S

索　引

索　引

参考文献・リンク

■JavaScript ゲームのための単純な 2D 物理エンジンを作成する

http://www.ibm.com/developerworks/jp/web/library/wa-build2dphysicsengine/

筆者には“目の肥えた読者には物理エンジン”という変な思い込みがあり、2D物理エンジンライブラリを検索していた時に見つけたサイトです。執筆に着手した当初は、自分でエンジンを作るなどまったく想像もしていませんでしたが、記事を読むにつれ“この程度なら自分で作れるかも”という思いになりました。この記事なくしてTiny2D.jsはありえませんでした。

■物理エンジン実装時に参考にしたリンク

http://kanasana.sblo.jp/article/28818677.html

http://hakuhin.jp/as/collide.html

http://wakariyasui.sakura.ne.jp/b2/52/5221unndouryouhozonn.html

http://www.geocities.jp/no_smoking_pool/study.html

http://marupeke296.com/COL_2D_No10_SegmentAndSegment.html

ベクトル演算の復習では上記サイトの情報を参考にさせていただきました。

■Kickass Java Programming: Cutting-Edge Java Techniques With an Attitude

ISBN-13: 978-1883577995

Java言語の本（英語）ですが、2D描画や3Dモデリングなどさまざまな技法が説明されている良書です。Saturn Voyagerはこの本のサンプルをヒントに実装しました。

■ゲームを作りながら楽しく学べるHTML5+CSS+JavaScriptプログラミング

（インプレスR&D刊 ペーパーバック版／電子書籍版）

本書の姉妹書とも言える拙著です。掲載しているゲームはすべて異なります。ほかのゲームやコードも見てみたいという方はご覧いただければ幸いです。

■BGM

http://takao-suenobu.com/

http://dova-s.jp/ というサイトからスエノブさんの素材をBGMとして利用させていただきました。ゲームの印象をガラッとかえることができました。

■その他（Reversiのアルゴリズム）

http://uguisu.skr.jp/othello/5-1.html

あ と が き

オバマ大統領はアメリカの若者にこう訴えかけました。

ゲームは買うだけでなく作ってみよう!
Don't just buy a new video game, make one.
スマホで遊ぶだけでなくプログラミングしよう!
Don't just play on your phone, program it.

また、Facebookの元役員のChamath Palihapitiyaはこう言ったそうです。

プログラミングを学ぶのなら、生涯仕事に困らないことを私が保証しよう

アメリカやイギリスは国を挙げてプログラミング教育に力を入れています。“国が発展していくためにはプログラミングスキルを持つ人材が必要だ”ということをトップが認識しているからにほかなりません。今後、プログラマー不足はますます深刻になると言われています。一流企業でも激しいリストラが行われる今日、プログラミングスキルを身に着けておくことは決して無駄にはならないはずです。幸いにも自分は世界中のDeveloperと一緒に製品を開発するという貴重な機会に恵まれましたが、これも自作のアプリが認められたことがきっかけでした。自分にプログラミングスキルが無かったら、そんな経験はできなかったことでしょう。

本書のとりあえずの目的は“自分でもゲームを作ってみたい”と興味を持ってもらうことです。「行数が少なくてわかりやすい」という制約の下、「市販ゲームで目の肥えた読者の興味をも惹きつける」という、一見すると矛盾する要件を満たすのは、かなりハードルの高いチャレンジでした。自分の目標がどこまで達成できたか不安な部分もあります。

“自分でもゲームを作ってみよう”と興味をもつ
→ ゲーム作りを通してプログラミングスキルを高めることができる
→ プログラミングスキルを持つことで将来の可能性が広がる

そんな若い読者が未来を切り拓くために、本書が一助になれば、このうえない喜びです。

最後にこの場を借りて感謝の意を表させてください。斜め向かいにお住いの井筒家とは長年家族ぐるみの付き合いをさせていただいていますが、井筒晴香さんにはFunky BlocksとVegetable Marchのデザインを担当していただきました。これら2つのゲームはほかのサンプルと比べて格段に良い仕上がりとなりましたが、これは晴香さんの尽力によるものです。卒業制作前の忙しい時期にも関わらず快く引き受けていただき、本当に感謝しております。

本書の刊行にあたり、ピーチプレス社の芹川 宏さんには自分の拙いWORD原稿を校正いただくとともに、書籍として素晴らしい体裁に仕上げていただきました。また「次の世代にプログラミングの面白さを伝えたい」と日頃から感じていた筆者に、このような執筆の機会を提供していただいたインプレスの方々にも深く感謝しております。

今年一年、本書の執筆のため長期休暇や週末などかなりの日数を犠牲にしてしまいました。文句ひとつ言わずサポートしてくれた家族にも感謝しています。ありがとうございました。

本書が皆様のお役に立つことを切に願っております。

2015年11月吉日　著者

著者プロフィール

田中賢一郎（たなかけんいちろう）

1994年慶應義塾大学大学院理工学部修了。同年キヤノン株式会社に入社。2000年にデジタル放送立ち上げの会社に出向。その間に一人でデータ放送ブラウザを実装し、マイクロソフトへソースライセンスする。2008年よりWindows Media Center TVチームの開発者としてマイクロソフト ディベロップメント株式会社へ。その後、Windows開発部、Xbox、Office 365と漂流し、2015年10月、中小企業診断士の登録を機にマイクロソフトを退社。現在はIT教育関連のキャリアを模索中。

趣味はジャズピアノ演奏。宮澤隆氏に師事。週末は横浜界隈のジャムセッションに出没。

診断365：http://shindan365.net

趣　味：https://www.youtube.com/user/kenchitanaka/videos

●本書の内容は、執筆時点（2015年11月現在）での情報をもとに書かれています。本書に掲載している手順や考え方は一例であり、すべての環境においてその手順や考え方が同様に行えることを保証するものではありません。本書の内容に関するご質問は、書名・ISBN(このページに記載)・お名前・電話番号と、該当するページや具体的な質問内容、お使いの動作環境などを明記のうえ、インプレスカスタマーセンターまでメールまたは封書にてお問い合わせください。なお、本書発行後に仕様が変更されたハードウェア、ソフトウェア、サービスの内容に関するご質問にはお答えできない場合があります。

また、以下のご質問にはお答えできませんのでご了承ください。

・書籍に掲載している手順以外のご質問

・ハードウェア、ソフトウェア、サービス自体の不具合に関するご質問

・インターネットや電子メール、固有のデータ作成方法に関するご質問

本書の利用によって生じる直接的または間接的な被害について、著者ならびに弊社では、一切の責任を負いかねます。

あらかじめご了承ください。

●落丁・乱丁本はお手数ですがインプレスカスタマーセンターまでお送りください。送料弊社負担にてお取り替えさせていただきます。但し、古書店で購入されたものについてはお取り替えできません。

■読者の窓口
インプレスカスタマーセンター
〒101-0051 東京都千代田区神田神保町一丁目105 番地
TEL　03-6837-5016　／　FAX　03-6837-5023
info@impress.co.jp

■書店／販売店のご注文窓口
株式会社インプレス 受注センター
TEL　048-449-8040
FAX　048-449-8041

ゲームで学ぶ（まなぶ）JavaScript（ジャバスクリプト）入門（にゅうもん）
—HTML5（エイチティーエムエル）&CSS（アンドシーエスエス）も身（み）につく!

2015年12月11日　初版発行

著　者　田中 賢一郎（たなか けんいちろう）

発行人　土田米一

発行所　株式会社インプレス
〒101-0051　東京都千代田区神田神保町一丁目105番地
TEL　03-6837-4635(出版営業統括部)
ホームページ　http://book.impress.co.jp/

印刷所　株式会社廣済堂

ISBN978-4-8443-3978-6　C3055

Printed in Japan

【CONTENTS】

9784844339786

1923055024007

ISBN978-4-8443-3978-6
C3055 ¥2400E

定価（本体2,400円＋税）

【本書のサンプルについて】タブレットおよびスマートフォンでは、OS/ブラウザによっては挙動にばらつきがあります。

物理エンジン付きだからアクション系ゲームも作れる!

物理エンジンのデモ画面

0001900
Score

※PC版の画面です

impress 株式会社インプレス